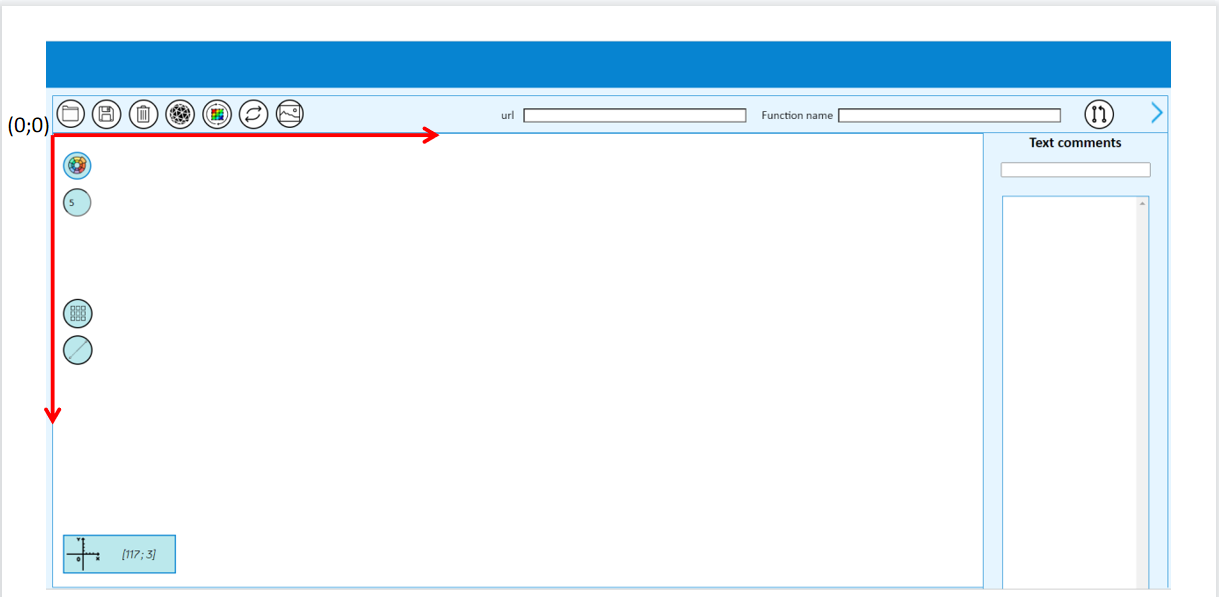
**THE WEB APP SIMULATION - SUMMARY**

1. **Event in Canvas**

Các sự kiện về event của con trỏ chuột trên Canvas được thiết lập trong class Paint.

Để bắt đầu vẽ các objects trên Canvas cần xác định các event của con chuột, các event gồm: mouse down, mouse move, mouse up.

Hàm lấy vị trí con trỏ chuột trên Canvas ***getMousePosition().*** Hàm này lấy gốc tọa độ của Canvas làm gốc tọa độ, mỗi đơn vị trên Canvas được tính là 1 pixel.



Các event của con trỏ chuột được khởi tạo từ đầu trong ***class Paint*** thông qua ***function listenEvent().***

Khởi tạo biến *isPainting* để xác định trạng thái vẽ cho Web App

1. ***Mouse down***

Vị trí mouse down được xác định khi click chuột trên Canvas và giá trị đó được lưu vào biến *mouseDownPos* thông qua hàm ***getMousePosition***

1. ***Mouse move***

Vị trí khi di chuyển con chuột trên Canvas cuungx được lưu lại vào biến *currentMouseMovePos* thông qua hàm ***getMousePosition***

1. ***Mouse up***

Vị trí con trỏ chuột được thả ra sau trạng thái mouseMove

1. **Các hàm vẽ trên Canvas**

***\* Hiển thị đường thẳng trên Canvas realtime khi khi dùng click và di chuyển chuột:***

Trong trạng thái isPainting, người dùng click chuột, Web App sẽ ghi nhận vị trí *mouseDown*, khi thả chuột Canvas nhận event *mouseMove* và thực hiện hàm ***drawLine*** từ vị trí *mouseDown* đến vị trí *mouseMovePos,* Web Appliên tục tạo Canvas mới bằng hàm ***undo*** để các đường thẳng được render một cách liên tục và không chồng chất lên nhau.

***\* Get nearest pixel***

Khi thực hiện vẽ hình học, Web App tự động truy bắt các điểm trên Canvas trong phạm vi 10 pixel xung quanh vị trí mouseMove/mouseDown thông qua hàm getNearest.

 getNearest(listPoints, currentPoint, maxDistance) {

    let distance = function (a, b) {

      return math.sqrt(Math.pow(a.x - b.x, 2) + Math.pow(a.y - b.y, 2));

    };

    let tree = new kdTree(listPoints, distance, ["x", "y"]);

    return tree.nearest(currentPoint, 1, maxDistance)[0];

  }

Hàm sử dụng thư viện kdTree để tìm điểm gần nhất với *currentPoint* trong list tất cả các điểm hiện có trên Canvas trong phạm vi *maxDistance* và trả về giá trị của điểm gần nhất [x,y].

***\* Thiết lập giá trị màu sắc và bề dày cho đường thẳng và điểm:***

Các giá trị màu sắc và bề dày nét vẽ được thiết lập trong hàm ***choiceEvent***, hàm này cũng được chạy từ lúc khởi tạo chương trình.

 choiceEvent() {

    this.toolbar.addEventListener("change", (e) => {

      if (e.target.id === "line\_color") {

        this.currentColor = e.target.value;

      }

      if (e.target.id === "line\_size") {

        this.currentWidth = e.target.value;

      }

      if (e.target.id === "sizeGrid") {

        if (this.currentValueGrid.value == "On") {

          this.deltaGrid = e.target.value;

          this.ctx.fillStyle = "white";

          this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);

          // function update object saved

          //redraw object

          this.renderObject(processingData.allLine);

          this.ctx.strokeStyle = "grey";

          this.drawGrid();

        }

      }

    });

  }

Khi cần lắng nghe sự kiện của 1 element nào đó trên Canvas cần gọi đúng id của element đó thông qua hàm ***addEvenListener(“elementID”, function)***

trong đó function là các lệnh cần thực hiện khi gọi đến event.

Trong web app, giá trị màu sắc và bề dày của các đối tượng lần lượt là *currentColor* và *currentWidth.* Giá trị mặc định của màu sắc và bề dày là “black” và 5:

this.currentColor = "black";

this.currentWidth = 5;

***\* Render objects***

Xuyên suốt quá trình hoạt động của Web App, hàm renderOject được dùng để đọc và hiển thị tất cả các thông tin của tất cả các objects (point, line, area).

1. **Processing data in Canvas**

Data hiện tại trên Canvas lưu trong thành các biến public, có 4 biến quan trọng nhất lưu tất cả các thông tin của objects:

processingData.allObject = []; //save all objects in Canvas

processingData.allLine = []; // save all Lines in Canvas

processingData.allPoint = []; //save all Points in Canvas

processingData.allArea = []; // save all areas in Canvas

1. ***Points***

Mỗi điểm trên Canvas được định nghĩa trong class Point:

Các thuộc tính:

    this.point = Arr; //coord

    this.x = Arr[0];

    this.y = Arr[1];

    this.className = "Point";

    this.name = pointName;

    this.force = []; // force in point

    this.moment = []; //moment

    this.pointLoads = pointLoads;

    this.soln = soln; //value solution in node

Các phương thức:

- Kiểm tra mouse click in point:

  isIn(mouse) {

    return (mouse[0] - this.x) \*\* 2 + (mouse[1] - this.y) \*\* 2 < 0.9 \* PaintIn.currentWidth \*\* 2

      ? true

      : false;

  }

Nếu giá trị mouse click nằm trên bán kính của điểm thì trả về giá trị true, ngược lại trả về false

- Kiểm tra điểm nằm trong select box (mouse move left to right):

  isInBox(topLeftPoint, bottomRigthPoint) {

    let point = this.point;

    if (

      topLeftPoint[0] < point[0] &&

      topLeftPoint[1] < point[1] &&

      point[0] < bottomRigthPoint[0] &&

      point[1] < bottomRigthPoint[1]

    ) {

      return true;

    } else return false;

  }

Nếu điểm nằm trong phạm vi của box giá trị trả về true, ngược lại trả về false

- Kiểm tra điểm nằm trong select box (mouse move right to left) *(đang phát triển)*

 isTouchBox(topRightPoint, bottomLeftPoint) {

    let point = this.point;

    if (topRightPoint != undefined &&

      topRightPoint[0] > point[0] &&

      topRightPoint[1] < point[1] &&

      point[0] > bottomLeftPoint[0] &&

      point[1] < bottomLeftPoint[1]

    ) {

      return true;

    } else return false;

  }

Nếu điểm chạm vào box hoặc nằm trong box thì giá trị trả về true, ngược lại false

1. ***Lines***

Mỗi đường thẳng trên Canvas được định nghĩa trong class Line:

Các thuộc tính:

    this.Point = [Point1, Point2]; //2 point start and end line

    this.color = lineColor;

    this.width = lineWidth;

    this.className = "Line";

    this.name = lineName;

    this.force = [];

    this.lineLoads = lineLoads;

    this.length = this.getLength();

this.bisectingPoint = this.getBisectingPoint();

Các phương thức:

- Tính chiều dài:

  getLength() {

    this.length = math.norm(

      math.subtract(this.Point[0].point, this.Point[1].point)

    );

  }

- Xác định trung điểm:

  getBisectingPoint() {

    this.bisectingPoint = [

      (this.Point[0].x + this.Point[1].x) / 2,

      (this.Point[0].y + this.Point[1].y) / 2,

    ];

  }

- Tạo các điểm trên line (hỗ trợ vẽ lực trên line)

getPointInLine(numPoint) {

    let numSeg = numPoint - 1;

    let point1 = this.Point[0].point;

    let point2 = this.Point[1].point;

    let dp = math.divide(math.subtract(point2, point1), numSeg);

    let subPoints = [];

    for (let i = 0; i < numPoint; i++) {

      let subPoint = math.add(point1, math.multiply(i, dp));

      subPoints.push(subPoint);

    }

    return subPoints;

  }

Hàm trả về list các điểm thuộc line với số lượng nút *numPoint* trên line xác định

- Tương tự point thì line cũng có các hàm kiểm tra con trỏ chuột lick vào vùng bề dày của line bằng hàm ***isIn*** và kiểm tra line nằm trong *box select* bằng hàm ***isInBox*** và hàm ***isTouchBox***

1. ***Areas***

Mỗi diện tích trên Canvas được định nghĩa trong class Area:

Các thuộc tính:

    this.className = "Area";

    this.Line = LineList; //line in area

    this.name = AreaName; // area name

    this.coordNaming = coorName; //coordinate input name

    this.pointFlow = this.getPointFlow();//point flow in an area

    this.perimeter = this.getPerimeter();

    this.area = this.getArea();

    this.center = this.getCenter();

Các phương thức:

- Tính chu vi: *getPerimeter()*

- Tính diện tích: getArea()

- Tính trọng tâm: getCenter()

- Xác định điểm tiếp theo để tạo diện tích kín:

getPointFlow() {

    this.pointFlow = [];

    for (let i = 0; i <= this.Line.length - 1; i++) {

      let Point1 = this.Line[i].Point[0].point;

      let Point2 = this.Line[i].Point[1].point;

      //

      if (i === this.Line.length - 1) {

        if (

          JSON.stringify(this.Line[0].Point).indexOf(JSON.stringify(Point2)) ===

          -1

        ) {

          let swap = Point1;

          Point1 = Point2;

          Point2 = swap;

        }

      } else {

        if (

          JSON.stringify(this.Line[i + 1].Point).indexOf(

            JSON.stringify(Point2)

          ) === -1

        ) {

          let swap = Point1;

          Point1 = Point2;

          Point2 = swap;

        }

      }

      this.pointFlow.push(Point1);

    }

  }

Thuật tóa kiểm tra điểm cuối của một line thuộc area trùng với điểm đầu của line tiếp theo và trả về list các điểm kế tiếp nhau tạo thành 1 area kín.

- Tương tự point và line, area cũng có các hàm kiểm tra con trỏ chuột lick vào vùng diện tích của area bằng hàm ***isIn*** và kiểm traarea nằm trong *box select* bằng hàm ***isInBox*** và hàm ***isTouchBox***

1. ***Hàm xử lý dữ liệu:***

RawData (mouseDown)

InputRawData

- Create Points

- Create Lines

- Area detected by OpenCV

- UpdateStorage

processingData.allPoint

processingData.allLine

processingData.allArea

processingData.allObject

Xử lý data được thực hiện trên class processingData.

Phương thức tạo dữ liệu cho object từ dữ liệu thô (vị trí *mouseDown*):

inputRawData(

    Type, //defined value with line/rect/circle/spl...

    Arr1, //list coordinate of x

    Arr2, //list coordinate of y

    listPointName, //list points name once render need to save

    listLineName, //list line name once render need to save

    colorList, //list corlor once render need to save

    widthList, //list width once render need to save

    arrForcePoint, //list force in point once render need to save

    arrForceLine //list force in line once render need to save

  )

Hiện tại web app chỉ mới xử lý dữ liệu cho trường hợp vẽ line, nên type = “line”

switch (Type) {

      case "line":

        {

          // create Point

          let AllPointObj = this.createPoint(

            Arr1,

            Arr2,

            listPointName,

            arrForcePoint

          );

          // create line

          let AllLineObj = this.createLine(

            AllPointObj,

            listLineName,

            colorList,

            widthList,

            arrForceLine

          );

List points được tạo từ hàm createPoint như sau:

 createPoint(arrPointX, arrPointY, nameList, pointLoadsList) {

    //tao point tu list toa do X va Y

    let AllPointObj = [];

    for (let index = 0; index <= arrPointX.length - 1; index++) {

      let point = [arrPointX[index], arrPointY[index]];

      let pointName = nameList[index];

      let pointLoads = pointLoadsList[index];

      let PointObj = new Point(point, pointName, pointLoads);

      AllPointObj.push(PointObj);

    }

    return AllPointObj;

  }

Đầu vào của hàm là list tọa độ theo x và y, list name của điểm và list lực đã có trên điểm. Từ các dữ liệu đó hàm tạo các điểm mới (new Point) và trả về list tất cả các điểm được tạo.

List lines được tạo từ hàm createLine như sau:

 createLine(PointList, nameList, colorList, widthList, lineLoadsList) {

    let AllLineObj = [];

    for (let index = 0; index <= PointList.length - 2; index++) {

      let Point1 = PointList[index];

      let Point2 = PointList[index + 1];

      let lineName = nameList[index];

      let lineColor = colorList[index];

      let lineWidth = widthList[index];

      let lineLoads = lineLoadsList[index];

      let LineObj = new Line(

        Point1,

        Point2,

        lineName,

        lineColor,

        lineWidth,

        lineLoads

      );

      AllLineObj.push(LineObj);

    }

    return AllLineObj;

  }

Đầu vào của hàm là list các điểm (2 point liên tiếp nhau tạo thành 1 line), list tên, màu sắc, bề dày và giá trị lực trên mỗi line.

Nếu tạo object đầu tiên thì các giá trị không có sẽ được thiieets lập mặc định là *undefined.*

Tiếp theo trong hàm inputRawData, thực hiện lưu points và lines vừa được tạo vào dữ liệu tổng thể:

for (let line of AllLineObj) {

            processingData.prototype.addObject(line, processingData.allLine);

}

Với:

addObject(newObject, saveArr) {

    let sameObj = saveArr.find(

      (value) => JSON.stringify(value) === JSON.stringify(newObject)

    ); // chuyen ve string de so sanh element trong array

    if (sameObj === undefined) {

      saveArr.push(newObject); //neu khong co phan tu tuong tu thi add phan tu do

    }

    return sameObj; // neu co phan tu tuong tu thi chi lay 1 phan tu

  }

Đầu vào của hàm ***addObject*** là *newObject* cần thêm vào mảng *saveArr* nếu *newObject* chưa tồn tại trong *saveArr* thì giá trị trả về mảng *saveArr* được thêm phần tử *newObject*, ngược lại trả về *newObject* đã tồn tại trong mảng.

Vì vậy thuật toán lưu giá trị lines vào dữ liệu tổng thể *processingData.allLine* sẽ không xuất hiện phần tử lặp.

Tiếp theo, thực hiện sắp xếp tất cả các dữ liệu từ point, line, area vào dữ liệu tổng thể thông qua hàm ***updateStorage()***. Hàm sắp xếp data theo tứ tự area => line => point.

1. **Features in Canvas**

\* Để bắt đầu thực hiện một chức năng nào đó trên Canvas cần ấn vào button của chức năng đó để set up cho web app biết trạng thái hiện tại.

1. **Grid**

Chức năng lưới trên Canvas được thiết lập ở hàm ***buttonGrid***, các nút của lưới được lưu trữ trong array *arrGrid*, số lượng nút phụ thuộc vào giá trị *deltaGrid* (giá trị mặc định của deltaGrid = 40 pixel).

Có thể thay đổi giá trị của *deltaGrid* bằng các event *keyDown*:

//KEYUP

if (event.keyCode === 38) {

      if (

        this.currentValueGrid.value == "On" &&

        this.deltaGrid <= this.maxGrid

      ) {

        this.deltaGrid += this.deltaGrid / 2;

//KEYDOWN

if (event.keyCode === 40) {

      if (

        this.currentValueGrid.value == "On" &&

        this.deltaGrid >= this.minGrid \* 2

      ) {

        this.deltaGrid -= this.deltaGrid / 2;

Với mỗi event keyup/keydown thì giá trị deltaGrid tăng/giảm đi một nửa giá trị.

1. **Import**

Chức năng import/open JSON file in Canvas được thiết lập trong hàm ***listenEvent***:

document.getElementById("openFile").addEventListener("change", function () {

      var fr = new FileReader();

      fr.onload = function () {

        let inputData = JSON.parse(fr.result);

        if (inputData["jsmat"] !== undefined) {

          Mesh.prototype.openFileSoln(inputData);

        } else {

          PaintIn.clearAll();

          processingData.prototype.createData(inputData);

          //update screen

          PaintIn.renderObject(processingData.allObject);

        }

      };

      fr.readAsText(this.files[0]);

    });

File được import và Canvas (inputData) được mở bằng viện **FileReader**(), nếu là file lưu trữ về geomertry thì giá trị *“jmat” === underfined* => hàm ***createData*** lấy các dữ liệu trong file input => tạo objects mới => render lên Canvas.

1. **Export/Save**

Tất các thuộc tính và giá trị của objects (points/lines/areas) được tổng hợp và lưu lại trong hàm ***saveObj*** (in class processingData) với format của JSON (list data).

Hàm saveAsData hỗ trợ export/save giá trị của các objects thành JSON file bằng thư viện **Blob.**

saveObj() {

    let data = JSON.stringify(processingData.allObject);

    let num\_nodes;

    let num\_segments;

    let nodes = [];

    let node\_names = [];

    let segments = [];

    let segment\_names = [];

    let surfaces = [];

    let surface\_names = [];

    let surface\_coords = [];

    let nodal\_loads = [];

    let segment\_loads = [];

    num\_nodes = processingData.allPoint.length;

    num\_segments = processingData.allLine.length;

    // //change coordinate Oxy origin

    // let rotMatrix = [

    //   [math.cos(math.PI / 2), -math.sin(math.PI / 2)],

    //   [math.sin(math.PI / 2), math.cos(math.PI / 2)]

    // ]

    // let allPoint = [...processingData.allPoint];

    // for (let i = 0; i <= allPoint.length - 1; i++) {

    //   let nodeCoord = [allPoint[i].x,allPoint[i].y];

    //   //rot

    //   nodeCoord = math.multiply(nodeCoord, rotMatrix);

    //   nodeCoord = nodeCoord.flat()

    //   allPoint[i].x = nodeCoord[0];

    //   allPoint[i].x = nodeCoord[1];

    // }

    // let allLine = [...processingData.allLine];

    // for (let i = 0; i <= allLine.length - 1; i++) {

    //   let startcoordX = allLine[i].Point[0].x

    //   let startcoordY = allLine[i].Point[0].y

    //   let endcoordX = allLine[i].Point[1].x

    //   let endcoordY = allLine[i].Point[1].y

    //   let startNode = [startcoordX,startcoordY];

    //   let endNode = [endcoordX,endcoordY];

    //   //rot

    //   startNode = math.multiply(startNode, rotMatrix);

    //   startNode = startNode.flat()

    //   endNode = math.multiply(endNode, rotMatrix);

    //   endNode = endNode.flat()

    //   allLine[i].Point[0].x = startNode[0];

    //   allLine[i].Point[0].y = startNode[1];

    //   allLine[i].Point[1].x = endNode[0];

    //   allLine[i].Point[1].x = endNode[1];

    // }

    for (let point of processingData.allPoint) {

      nodes.push(point.point);

      node\_names.push(point.name);

      nodal\_loads.push(point.pointLoads);

    }

    for (let line of processingData.allLine) {

      let index1 = nodes.findIndex(

        (value) => JSON.stringify(value) === JSON.stringify(line.Point[0].point)

      );

      let index2 = nodes.findIndex(

        (value) => JSON.stringify(value) === JSON.stringify(line.Point[1].point)

      );

      let segment = [index1, index2];

      segments.push(segment);

      segment\_names.push(line.name);

      segment\_loads.push(line.lineLoads);

    }

    for (let area of processingData.allArea) {

      let surface = [];

      for (let i = 0; i <= area.pointFlow.length - 1; i++) {

        let point = area.pointFlow[i];

        let index = nodes.findIndex(

          (value) => JSON.stringify(value) === JSON.stringify(point)

        );

        surface.push(index);

      }

      surfaces.push(surface);

      surface\_names.push(area.name);

      surface\_coords.push(area.coordNaming);

    }

    let jsonObject = {

      num\_nodes: num\_nodes,

      num\_segments: num\_segments,

      node\_coords: nodes,

      node\_names: node\_names,

      segments: segments,

      segment\_names: segment\_names,

      surfaces: surfaces,

      surface\_names: surface\_names,

      surface\_coords: surface\_coords,

      nodal\_loads: nodal\_loads,

      segment\_loads: segment\_loads,

      text\_data: dataLogFile,

    };

    let dataSaved = JSON.stringify(jsonObject);

    return dataSaved;

  }

  saveAsData() {

    let jsonData = processingData.prototype.saveObj();

    let blob = new Blob([jsonData], { type: "text/plain;charset=utf-8" });

    saveAs(blob, "data.json");

  }

* Hiện tại toạ độ của objects lưu theo hệ toạ độ của Canvas, có thể thay đổi giá trị lưu về hệ toạ độ quy định (có thể tham khảo phần comments trong funciton).

1. **New / Clear All**

Hàm ***clearAll*** (in calss Paint) tạo mới/ xoá toàn bộ dữ liệu đã có trên Canvas và trả lại trạng thái mặc định của Canvas (select):

* Tạo canvas mới

this.ctx.fillStyle = "white";

this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);

* Chuyển con trỏ chuột về dạng select

this.pen = "select";

this.currentCursor = "url(frontend/img/select\_cursor.svg) 0 0,  default";

this.canvas.style.cursor = this.currentCursor;

* Xoá tất cả các dữ liệu

this.arrMouseDownPosition = [];

    this.curSelectBox = [];

    this.arrLineX = [];

    this.arrLineY = [];

    this.arrCircleX = [];

    this.arrCircleY = [];

    this.arrRectX = [];

    this.arrRectY = [];

    this.arrLineColor = [];

    this.arrLineWidth = [];

  processingData.allLine = [];

    processingData.allPoint = [];

    processingData.allArea = [];

    processingData.allObject = [];

    this.arrCurObj = [];

    this.arrMultiCurObj = [];

* Tắt tất cả command và các chức nnagw khác nếu chúng đang hoạt động:

this.renderProperty("off", "");

    this.renderObject(processingData.allObject);

    PaintIn.clearCommands("textCommands");

+ Tắt trạng thái vẽ (nếu trạng thái vẽ vẫn đang hoạt động):

this.isCancled = false;

this.offButtonDraw(this.currentValueLine, "line");

+ Tắt chế độ lưới (nếu lưới mở)

 if (this.currentValueGrid.value == "On") {

      this.ctx.strokeStyle = "grey";

      this.drawGrid();

    }

1. **Meshing**

* ***Mesh lưới tam giác bậc cao (Matlab server online)***

Hiện tại chưa kết nối và gửi trực tiếp data từ web app đến Matlab Server online, class Mesh (in Mesh.js file) đọc giá trị trong file solution và hiển thị kết quả lên Canvas.

Mở file kết quả thông qua chức năng ***import*** của Canvas:

document.getElementById("openFile").addEventListener("change", function () {

      var fr = new FileReader();

      fr.onload = function () {

        let inputData = JSON.parse(fr.result);

        if (inputData["jsmat"] !== undefined) { //open file solution

          Mesh.prototype.openFileSoln(inputData);

        } else {

          PaintIn.clearAll();

          processingData.prototype.createData(inputData);  //open data in Canvas

          //update screen

          PaintIn.renderObject(processingData.allObject);

        }

      };

      fr.readAsText(this.files[0]);

    });

Giá trị “jmat” dùng để phân biệt file solution và file data trên Canvas.

Đọc data từ file soluction:

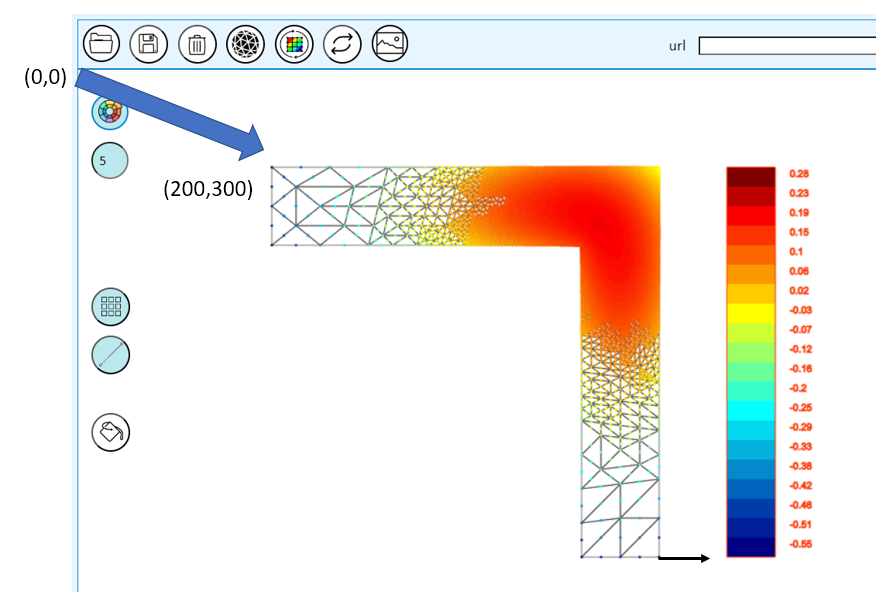
Hàm createDataMesh đọc giá trị của nodes/edges/surfaces trả về:

* “jsmat”: chứa thông tin dữ liệu của geometry:
* “num\_nodes”: số lượng points trên canvas tạo thành geometry
* “num\_segments”: số lượng lines
* “node\_coords”: toạ độ points
* “node\_names”: point names
* “segments”: thứ tự points tạo thành segments (1 segment được tạo thành từ 2 points)
* “segment\_names”: line names
* “surfaces”: thứ tự points tạo thành surfaces
* “surface\_names”: area names
* “nodal\_loads”: giá trị lực apply trên points
* “segment\_loads”: giá trị lực trên lines
* “Fetri": thứ tự nodes trên lưới tạo thành element, mỗi vị trí có giá trị tương ứng với index trong “Fecoord”.
* “Fecoord”: toạ độ của nodes trên lưới (toạ độ đã được scale theo tỉ lệ chiều dài và rộng của geometry)

A picture containing chart

Description automatically generated

* “Fesoln”: giá trị lực sau khi giải, mỗi giá trị ứng với index của mỗi nút của “Fetri”.
* if (Mesh.inputData["QC"].length !== 0) {
* jsmat = Mesh.inputData["jsmat"];
* FEsoln = Mesh.inputData["FEsoln"];
* QC = Mesh.inputData["QC"];
* baseCoord = jsmat["node\_coords"][3];
* baseCoord = [200, 300];
* FEtri = Mesh.inputData["FEtri"];
* FEcoord = Mesh.inputData["FEcoord"];
* scale = 400;
* let arrNodeColor = [];
* let arrElem = [];
* const FEcoordNode = JSON.parse(JSON.stringify(FEcoord));
* const FEcoordLine = JSON.parse(JSON.stringify(FEcoord));
* let indexValueColor = [];
* let colors = colormap({
* colormap: "jet",
* nshades: 100,
* format: "hex",
* alpha: 2,
* });
* *scale = 400* (giá trị chiều dài của segments trong file kết quả (Uip\_FE.json), giá trị này cần được thay đổi theo chiều dài của từng geometry với từng bài khác nhau.
* *baseCoord = [200,300]:* move hệ toạ độ render hình từ vị trí gốc toạ độ của Canvas đến vị trí [200,300] giúp quá trình hiển thị trực quan hơn.



Chế độ color bar:

Shape

Description automatically generated

Button fillColor chuyển trạng thái từ vẽ từng phần tử sang fill color bar, hàm fillElements đọc giá trị của “Fesoln” => dựa vào các giá trị, quy đổi, set value cho dãy màu từ giá trị min đến giá trị max của “Fesoln”.

* ***Mesh lưới tam giác (test version in local Matlab server (thuật toán mesh của anh Phương)):***

Version này cũng được thiết lập trong class Mesh với giá trị “QC” == undefined, version test quá trình gửi data trực tiếp từ web app đến local Matlab server.

Thuật toán mesh lưới tam giác nhận input data là giá trị toạ độ của geometry trên Canvas và data request gửi cho Matlab server thông qua hàm ***mps\_PALc,*** sử dụng thư viện **axios** với phương thức POST để gửi data cho local Matlab Server.

 mps\_PALc(pname, params) {

    //bodyData: data will send to server

    //params: list data - JSON form

    // params = {"num\_nodes":6,"num\_segments":6,"node\_coords":[[280,200],[280,340],[460,340],[460,420],[580,420],[580,120]],"node\_names":[null,null,null,null,null,null],"segments":[[0,1],[1,2],[2,3],[3,4],[4,5],[0,5]],"segment\_names":[null,"SeA","SeB",null,null,"SeC"],"surfaces":[],"surface\_names":[],"nodal\_loads":[null,null,null,null,null,null],"segment\_loads":[null,null,null,null,null,null],"text-data":["","",""]}

    let bodyData = {

      rhs: [pname, params], //rhs: reading - used when send data

      nargout: 1,

      outputFormat: { mode: "small", nanType: "object" },

    };

    let promise = axios({

      headers: {

        "Content-Type": "application/json",

      },

      method: "POST",

      // url: "http://localhost:5902/matfun/mps\_PAL",

      url: "http://localhost:9910/bondTools/firstAPI",

      url: urlSendRequest,

      data: bodyData,

    });

    promise.then((result) => {

      let receiveData = result.data["lhs"][0];

      Mesh.prototype.openFileSoln(receiveData);

      if (receiveData !== undefined) {

        dataLogFile.push(JSON.stringify(receiveData));

        PaintIn.renderCommand("textCommands");

      }

    });

    promise.catch(function (err) {

      console.log("err", err);

      dataLogFile.push(JSON.stringify(err));

      PaintIn.renderCommand("textCommands");

    });

  }

Với kết quả local Matlab Server trả về, hàm ***openFileSoln*** đọc kết quả (kết quả của hàm trả về là giá trị toạ độ của các node) và thực hiện render các giá trị lên Canvas.

* ***Chuyển đổi mode solution và mode drawing bằng hàm changeMode***

changeMode() {

    if (this.curValDrawing.value === "Off") {

      // mode drawing

      this.renderObject(processingData.allObject);

      this.curValDrawing.value = "On";

      document.getElementById("modeDrawing").classList.add("active");

      Mesh.curValFillColor.value = "Off";

      document.getElementById("fillColor").style.display = "none";

      this.mouseMoveStatus = true;

      this.pen = "select";

      this.curValSelect = "On";

    } else {

      //mode soln

      this.currentCursor = "url(frontend/img/select\_cursor.svg) 0 0,  default";

      this.canvas.style.cursor = this.currentCursor;

      this.ctx.fillStyle = "white";

      this.ctx.fillRect(0, 0, this.canvas.width, this.canvas.height);

      this.curValDrawing.value = "Off";

      document.getElementById("modeDrawing").classList.remove("active");

      Mesh.curValFillColor.value = "Off";

      document.getElementById("fillColor").classList.remove("active");

      this.mouseMoveStatus = false;

      this.pen = undefined;

      this.curValSelect = "Off";

      //display soln

      if (Mesh.inputData !== undefined) {

        document.getElementById("command").style.display = "none";

        Mesh.prototype.drawMesh();

      } else {

        this.renderCommand("soln");

      }

    }

  }

1. **Detect pictures**

Button import hình ảnh vào web app:



Dữ liệu hình ảnh input vào web app được xử lý và image data được thiết lập ở hàm listenEvent:

//input img

    let form = document.getElementById("inputImg");

    form.addEventListener("change", function (event) {

      event.preventDefault();

      const formData = new FormData(form[0]);

      formData.append("file", $("#inputImg")[0].files[0]);

      let promise = axios({

        method: "POST",

        url: "https://vysecondapp.herokuapp.com/v1/picture/",

        data: formData,

        headers: {

          "Content-Type": "multipart/form-data",

        },

      });

      promise.then((result) => {

        processingData.prototype.createData(result.data);

        //update screen

        PaintIn.renderObject(processingData.allObject);

      });

      promise.catch(function (err) {

        console.log("err", err);

      });

    });

Image data => xử lý và đưa về formData => sử dụng thư viện axios POST data cho openCV (in backend) xử lý dữ liệu => gửi data về web app render lên Canvas.

1. **Naming to create lable for point/line/area**

Sử dụng thư viện CanvasInput => input name for point/line/area. Hàm inputName/inputNames (in inputData.js) dùng để đặt tên/tạo lable.

Có 2 trường hợp input name: inputname cho 1 đối tượng và inputNames cho nhiều đối tượng cùng lúc.

Cần setup vị trí hiển thị ô input trước khi sử dụng thư viện CanvasInput để đặt tên cho object. Hàm setup vị trí đặt tên addName (in addValue.js file).

Trước tên cần kiểm tra các điều kiện của ô input tránh bị chồng chất sự kiện trên Canvas:

//check before input

    PaintIn.isMovingObj = false;

    if (PaintIn.arrMultiCurObj[0] !== undefined) {

        if (valueLoads !== undefined && PaintIn.curValPressLoad.value === "Off") {

            valueLoads.destroy();

            valueLoads = undefined;

        }

        if (valueMoments !== undefined && PaintIn.curValMoment.value === "Off") {

            valueMoments.destroy();

            valueMoments = undefined;

        }

    }

    if (PaintIn.arrCurObj[0] !== undefined) {

        //name on=> press & moment off

        if (valueLoad !== undefined && PaintIn.curValPressLoad.value === "Off") {

            valueLoad.destroy();

            valueLoad = undefined;

        }

        if (valueMoment !== undefined && PaintIn.curValMoment.value === "Off") {

            valueMoment.destroy();

            valueMoment = undefined;

        }

    }

Ứng với mỗi loại object sẽ có các hàm set up vị trí riêng:

Set up vị trí ô input tên trên point:

function addNamePoint(Obj) {

    let pos = getPosElement("valueName");

    if (PaintIn.arrMultiCurObj[0] !== undefined) {

        inputNames(pos[0] + 15, pos[1]);

        return;

    } else {

        let xC = Obj.x;

        let yC = Obj.y;

        inputName(xC, yC, Obj);

        return;

    }

}

Hàm getPosElement (in painting.js file): set up vị trí của ô input trong trường hợp naming cho nhiều đối tượng cùng lúc.

function getPosElement(idElem) {

  let elem = document.getElementById(idElem);

  let left = elem.getBoundingClientRect().left;

  let top = elem.getBoundingClientRect().top;

  let right = elem.getBoundingClientRect().right;

  let bottom = elem.getBoundingClientRect().bottom;

  //- padding

  let xC = left - (right - left) / 2;

  let yC = top + (bottom - top) / 2 - toolTopHeight + 30;

  return [xC, yC];

}

Set up vị trí ô input tên cho line:

function addNameLine(Obj) {

    let pos = getPosElement("valueName");

    if (PaintIn.arrMultiCurObj[0] !== undefined) {

        inputNames(pos[0] + 15, pos[1]);

        return;

    } else {

        //choose position to display box input

        let xM1 = (Obj.Point[1].x - Obj.Point[0].x) / 2;

        let yM1 = (Obj.Point[1].y - Obj.Point[0].y) / 2;

        let xBox = 25 / 2;

        let yBox = 25 / 2;

        let xM2 = Obj.Point[0].x + xM1 - xBox;

        let yM2 = Obj.Point[0].y + yM1 - yBox;

        inputName(xM2, yM2, Obj);

        return;

    }

}

Set up vị trí ô input tên cho area:

function addNameArea(Obj) {

    //    choose position to display box input

    if (PaintIn.arrMultiCurObj[0] !== undefined) {

        let pos = getPosElement("valueName");

        inputNames(pos[0] + 15, pos[1]);

        return;

    } else {

        // let xC = Obj.center[0];

        // let yC = Obj.center[1];

        let xC = PaintIn.mouseDownPos.x;

        let yC = PaintIn.mouseDownPos.y;

        inputName(xC, yC, Obj);

        return;

    }

}

Tên của area được render đúng tại vị trí mouseDown select area.

**Hàm input name trên Canvas:**

* ***TH1: InputName****:* Naming cho 1 đối tượng (point/line/area).

+ Set up size/font cho ô input:

canvas: document.getElementById("myCanvas"),

    x: x,

    y: y,

    fontSize: 18,

    fontFamily: "Arial",

    fontColor: "#212121",

    fontWeight: "bold",

    width: 45,

    height: 25,

    padding: 0,

    borderColor: "#000",

    borderRadius: 3,

Hàm onsubmit: lưu giá trị nhập vào ô input vào đối tượng được chọn sau khi nhấn Enter.

onsubmit: function () {

      PaintIn.drawText(obj, this.value());

      obj.name = this.value();

      if (obj.className === "Area") {

        const coorName = [PaintIn.mouseDownPos.x, PaintIn.mouseDownPos.y];

        obj.coordNaming = coorName;

      }

      this.destroy(); //off text input

      nameID = undefined;

      PaintIn.renderObject(processingData.allObject);

      PaintIn.isCancled = false;

    },

* Nếu object được chọ là 1 area, cần lưu thêm vị trí con trỏ chuột (noi được chọ để input giá trị)

if (obj.className === "Area") {

        const coorName = [PaintIn.mouseDownPos.x, PaintIn.mouseDownPos.y];

        obj.coordNaming = coorName;

      }

* *focus: hàm giúp focus vào ô input ngay khi nhấn button inputName (nếu không cần phải click chuột vào ô input để thực hiện input name)*

nameID.focus();

* ***TH2: InputNames:*** chọn nhiều đối tượng cùng loại và naming chúng cùng lúc. Thuật toán tương tự như naming cho 1 đối tượng cụ thể.

Hàm renderObject đọc giá trị tên của objects => gọi hàm ***drawText*** (in class Paint) render object names.

1. **Add/deleted force (in point and line)**

Tương tự input name, add force in points/lines cũng sử dụng thư viện CanvasInput để nhập giá trị và có 2 trường hợp add force cho 1 point/line (inputForce) và add force cho nhiều points/line cùng lúc (inputForces).

Set up vị trí hiển thị ô input trong hàm addForce tương tự như input name.

Xác định loadKey cho mỗi object:

* Points: loadKey = “force”: lực tập trung tại điểm theo phương x và y
* Lines: loadKey = “normal\_pressure”

Lực trên point gồm lực theo phương x và y, hiển thị cùng ô input là hướng dẫn cách nhập giá trị cho người dùng được thiết lập trong hàm addCommand.

Giá trị nhập vào trong ô input cần được sử lý đưa về số:

force\_x = Number(this.value().slice(0, this.value().indexOf(",")));

force\_y = Number(

            this.value().slice(

              this.value().indexOf(",") + 1,

              this.value().length

            )

Giá trị nhập vào được lưu trực tiếp vào thuộc tính của object

forceObj = {

          type: loadKey,

          parameters: { force\_x: force\_x, force\_y: force\_y },

        };

        obj.pointLoads.push(forceObj);

Tương tự như lực trên point, giá trị lực input vào line cũng được xử lý về dạng số và lưu trữ vào các thuộc tính tương ứng của objects.

Hàm renderObject đọc giá trị lực của objects => ứng với mỗi loại objects sẽ có hàm vẽ lực khác nhau:

* Point: ***drawForceInPoint***
* Line: ***drawPressure***
* AddMoment:

Button apply giá trị moment cho điểm:



Apply moment vào point cũng có 2 trường hợp là apply cho 1 point (inputMoment) và apply cho nhiều points (inputMoments)

*loadKey = “moment”;*

Vị trí input moment giống với vị trí input lực trên point. Hàm vẽ moment trên điểm ***drawMoment*** (in class Paint).

* Setup position for BDcondition

Size, hoạt động của các element chứa các button addName, addForce, addMoment thiết lập ở phần renderObject:

// render properties and button

    if (this.arrMultiCurObj[0] !== undefined) {

      document.getElementById("BDCondition").style.display = "flex";

      this.renderProperty("multi", this.arrMultiCurObj);

      for (let i = 0; i < this.arrMultiCurObj.length; i++) {

        let selectedObj;

        selectedObj = this.arrMultiCurObj[i];

        switch (selectedObj.className) {

          case "Point":

            this.drawPoint(selectedObj, "green");

            document.getElementById("BDCondition").style.width = "200px";

            //display 3 button

            this.visibleButton("valueName");

            this.visibleButton("pointLoad");

            this.visibleButton("moment");

            //hidden 1 button

            this.hiddenButton("pressLoad");

            break;

          case "Line":

            this.drawLine(

              selectedObj.Point[0],

              selectedObj.Point[1],

              "#0000ff",

              selectedObj.width

            );

            document.getElementById("BDCondition").style.width = "150px";

            //display 2 button

            this.visibleButton("valueName");

            this.visibleButton("pressLoad");

            //hidden 2 button

            this.hiddenButton("pointLoad");

            this.hiddenButton("moment");

            break;

          case "Area":

            this.fillArea(selectedObj, "#b6d8e7");

            document.getElementById("BDCondition").style.width = "70px";

            //display 1 button

            this.visibleButton("valueName");

            //hidden 3 button

            this.hiddenButton("pressLoad");

            this.hiddenButton("pointLoad");

            this.hiddenButton("moment");

            break;

        }

      }

    } else {

      document.getElementById("BDCondition").style.display = "none";

      this.renderProperty("off", this.arrMultiCurObj);

    }

    if (this.arrCurObj[0] !== undefined) {

      let selectedObj;

      selectedObj = this.arrCurObj[0];

      switch (selectedObj.className) {

        case "Point":

          document.getElementById("BDCondition").style.display = "flex";

          document.getElementById("BDCondition").style.width = "200px";

          this.drawPoint(selectedObj, "green");

          this.renderProperty("point", selectedObj);

          //display 3 button

          this.visibleButton("valueName");

          this.visibleButton("pointLoad");

          this.visibleButton("moment");

          //hidden 1 button

          this.hiddenButton("pressLoad");

          break;

        case "Line":

          document.getElementById("BDCondition").style.width = "150px";

          document.getElementById("BDCondition").style.display = "flex";

          this.drawLine(

            selectedObj.Point[0],

            selectedObj.Point[1],

            "#0000ff",

            selectedObj.width

          );

          this.renderProperty("line", selectedObj);

          //display 2 button

          this.visibleButton("valueName");

          this.visibleButton("pressLoad");

          //hidden 2 button

          this.hiddenButton("pointLoad");

          this.hiddenButton("moment");

          break;

        case "Area":

          document.getElementById("BDCondition").style.width = "70px";

          document.getElementById("BDCondition").style.display = "flex";

          this.fillArea(selectedObj, "#b6d8e7");

          this.renderProperty("area", selectedObj);

          //display 1 button

          this.visibleButton("valueName");

          //hidden 3 button

          this.hiddenButton("pressLoad");

          this.hiddenButton("pointLoad");

          this.hiddenButton("moment");

          break;

      }

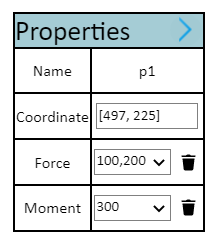
    }

Với từng điều kiện, sử dụng 2 hàm hiddenButton và visibleButton để ẩn/hiện các button trên Canvas.

Hàm hiddenButton => ẩn button khỏi canvas

Hàm visibleButton => hiện button lên Canvas

* ***Xoá lực và moment:***



Click chọn object cần xoá lực/moment => click vào biểu tượng xoá trong bảng propertiy

Hàm xoá lực:

delLoad(tagId) {

    let obj = this.arrCurObj[0];

    let selectTag = document.getElementById(tagId);

    let selectedLoad = selectTag[selectTag.selectedIndex];

    if (selectedLoad !== undefined) {

      if (obj.className === "Point") {

        obj.pointLoads.splice(selectedLoad.value, 1);

      } else if (obj.className === "Line") {

        obj.lineLoads.splice(selectedLoad.value, 1);

      }

    }

    this.renderObject(processingData.allObject);

  }

Thay đổi giá trị lực: nhập giá trị mới vào ô input tương ứng trong bảng property

Hàm thay đổi lực:

changeLoad(loadIndex, newValue, type) {

    switch (type) {

      case "force": {

        if (!newValue.includes(",")) return;

        let force\_x = Number(newValue.slice(0, newValue.indexOf(",")));

        let force\_y = Number(

          newValue.slice(newValue.indexOf(",") + 1, newValue.length)

        );

        //check fail input

        if (isNaN(force\_x) || isNaN(force\_y)) return;

        //set new value

        this.arrCurObj[0].pointLoads[loadIndex].parameters.force\_x = force\_x;

        this.arrCurObj[0].pointLoads[loadIndex].parameters.force\_y = force\_y;

        break;

      }

      case "moment": {

        let value = Number(newValue);

        //check fail input

        if (isNaN(value)) return;

        this.arrCurObj[0].pointLoads[loadIndex].parameters.value = value;

        break;

      }

      case "normalPress": {

        let node\_0 = Number(newValue.slice(0, newValue.indexOf(",")));

        let node\_1 = Number(

          newValue.slice(newValue.indexOf(",") + 1, newValue.length)

        );

        //check fail input

        if (isNaN(node\_0) || isNaN(node\_1)) return;

        //set new value

        this.arrCurObj[0].lineLoads[loadIndex].parameters.node\_0 = node\_0;

        this.arrCurObj[0].lineLoads[loadIndex].parameters.node\_1 = node\_1;

        break;

      }

    }

    this.renderObject(processingData.allObject);

  }

Giá trị lực/moment cũ của object được thay thế bằng giá trị mới input vào (chỉ thay thế khi người dùng nhập đúng cú pháp).

1. Display property table

Property table trình bày các thuộc tính hiện có của object khi object đó được chọn. Mỗi mode khác nhau ứng với từng trạng thái và loại object, property table catch từng trường hợp và render các thuộc tính cần thiết.

Với mỗi mode, property table có số hàng và cột khác nhau => thực hiện điều chỉnh css trong file javascript trong hàm ***renderProperty***:

 document.getElementById("property").innerHTML = `

        <div class="property\_label">

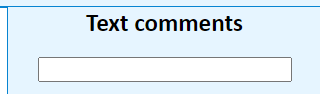
        <p>Properties</p>

        <div>

`

1. **Text comment** (input string to naming/add force for these elements which had name) and display the result when interaction about server and the web app

Text comments là khu vực hiển thị giá trị người dùng nhập vào ô input (*valueComment*)



và giá trị trả về khi kết nối với Matlab server, các giá trị này được lưu vào biến *dataLogFile.*

Hàm hiển thị giá trị text comment:

***renderCommand(mode)***, với mode = “textCommands”

case "textCommands":

        //render Text commands

        dataLogFileIndex = dataLogFile.length - 1;

        let strings = "";

        let reverseData = [...dataLogFile].reverse();

        for (let i in reverseData) {

          strings += reverseData[i] + "<br>";

        }

        document.getElementById("valueInputed").innerHTML = `

      <p style="background-color: #ffffff;

      height: 100%;

      overflow: scroll;

      border: 1px solid #0784d1;

      user-select: text;"> ${strings}</p>

      `;

Chuỗi strings xử lý tách chuỗi và sắp xếp giá trị trong dataLogFile => hiển thị giá trị theo thứ tự thời gian người dùng nhập vào.

1. **Connect Matlab Server (getAPI)**

Function thiết lập quá trình kết nối giữa Web app và Matlab Server bằng cách nhập url và tên subfuncton:

testAPI() {

      // show spinner

    if (PaintIn.APIurl.value !== "") {

      document.getElementById("spinner").style.display = "flex";

      urlSendRequest = PaintIn.APIurl.value;

      let pname = PaintIn.urlFunc.value;

      let params = processingData.prototype.saveObj();

      let bodyData = {

        rhs: [pname, params], //rhs: reading - used when send data

        nargout: 1,

        outputFormat: { mode: "small", nanType: "object" },

      };

      let promise = axios({

        headers: {

          "Content-Type": "application/json",

        },

        method: "POST",

        url: urlSendRequest,

        data: bodyData

      });

      promise.then((result) => {

        document.getElementById("spinner").style.display = "none";

        let receiveData;

        try {

          receiveData = result.data["lhs"][0];

          if (receiveData !== undefined) {

            dataLogFile.push(JSON.stringify(receiveData));

            PaintIn.renderCommand("textCommands");

          }

        } catch (err) {

          console.log(result);

          dataLogFile.push(JSON.stringify(result.data));

          PaintIn.renderCommand("textCommands");

        }

      });

      promise.catch(function (err) {

        document.getElementById("spinner").style.display = "none";

        console.log("err", err);

        dataLogFile.push(JSON.stringify(err));

        PaintIn.renderCommand("textCommands");

      });

//      spinner.style.display = "none";

    }

  }

*urlSendRequest*: địa chỉ url của Matlab Server

*pname*: tên subFunction Matlab Server

*params*: list data lưu thông tin của tất cả các objects trên Canvas dưới dạng JSON

Vì gửi data cho Matlab Server nên phần bodyData cần thiết lập thêm thông số: + “rhs” chứa list data

+ “nargout”

+ “outputFormat”

Sử dụng thư viện axios kết nối với Matlab Server bằng url

Kết quả Matlab Server trả về được lưu vào biến dataLogFile và render lên khu vực text area.

1. **Select objects**

Hàm selectObject thực thiện chức năng select các đối tượng (points/ lines/ areas).

Có 2 trường hợp select:

* Bounding box:

Draw bounding box in mouseMove function:

//bounding box

    if (

      this.isPainting &&

      !this.isMovingObj &&

      (this.pen === undefined || this.pen === "select") &&

      this.curValSelect === "On"

    ) {

      //draw bounding box

      this.undo();

      this.ctx.beginPath();

      this.ctx.fillStyle = "rgb(0 234 255 / 26%)";

      let topLeftPoint = [this.mouseDownPos.x, this.mouseDownPos.y];

      let bottomRigthPoint = [mouseMovePos.x, mouseMovePos.y];

      //save select box size

      this.curSelectBox = [topLeftPoint, bottomRigthPoint];

      this.ctx.fillRect(

        topLeftPoint[0],

        topLeftPoint[1],

        bottomRigthPoint[0] - topLeftPoint[0],

        bottomRigthPoint[1] - topLeftPoint[1]

      );

      //save the last mouse move

      this.lastMouseMove = bottomRigthPoint;

Check điểm nằm trong bounding box: Mỗi object đều có phương thức isInBox => sử dụng phương thức ***isInBox*** kiểm tra điểm nằm trong box select.

Các object thỏa điều kiện nằm trong bounding box => lưu vào *PaintIn.arrMultiCurObj.*

* Mouse click
* Single: phương thức ***isIn*** của mỗi object => kiểm tra mouse click trên object. Object thỏa điều kiện isIn được lưu vào arrCurObj.
* Multiple: điều kiện thực hiện chức năng multiSelect:

if (event.ctrlKey)

Sử dụng phương thức isIn của mỗi object => kiểm tra mouse click => các objects thỏa mãn điều kiện => lưu vào arrMultiCurObj

Các objects được chọn trong arrCurObj/arrMultiCurObj => hàm ***renderObject*** => đổi màu tất cả các objects đang được chọn.

1. **Delete objects**

Hàm xóa các objects ***deleteCurObj***, hàm xóa tất cả các objects đang được chọn trong arrCurObj/ arrMultiCurObj.

Hàm phân loại các objects dựa vào className => tìm vị trí object trong arrCurObj/ arrMultiCurObj trong list chứa tổng các objects cùng loại => xóa phần tử đó => update dữ liệu => render dữ liệu mới lên Canvas:

for (let Obj of this.arrCurObj) {

      if (Obj.className === "Point") {

        processingData.allPoint.splice(processingData.allPoint.indexOf(Obj), 1); //delete in allPoint

      } else if (Obj.className === "Line") {

        //if line in multi area

        let areaDels = [];

        processingData.allArea.forEach((area) => {

          for (let line of area.Line) {

            if (JSON.stringify(line) === JSON.stringify(Obj)) {

              areaDels.push(area);

            }

          }

        });

        //delete multi area

        for (let area of areaDels) {

          processingData.allArea.splice(

            processingData.allArea.indexOf(area),

            1

          ); //delete in allArea

        }

        processingData.allLine.splice(processingData.allLine.indexOf(Obj), 1); //delete in allLine

      } else if (Obj.className === "Area") {

        processingData.allArea.splice(processingData.allArea.indexOf(Obj), 1); //delete in allArea

      }

    }

    this.arrCurObj = [];

    for (let Obj of this.arrMultiCurObj) {

      if (Obj.className === "Point") {

        processingData.allPoint.splice(processingData.allPoint.indexOf(Obj), 1); //delete in allPoint

      } else if (Obj.className === "Line") {

        processingData.allLine.splice(processingData.allLine.indexOf(Obj), 1); //delete in allLine

      } else if (Obj.className === "Area") {

        processingData.allArea.splice(processingData.allArea.indexOf(Obj), 1); //delete in allArea

      }

    }

    this.arrMultiCurObj = [];

Update dữ liệu và render giá trị mới cập nhật lên Canvas:

//update storage

    processingData.prototype.updateStorage();

    //update screen

    this.renderObject(processingData.allObject);

1. **Move objects**

Hàm moveObject (in processingData class) có chức năng thiết lập di chuyển cho từng object trên Canvas.

Với từng đối tượng khác nhau sẽ có thuật toán di chuyển khác nhau:

Object cần di chuyển được lưu trong arrCurObj[0], mỗi lần chỉ di chuyển được 1 object.

* ***Point:***

Obj selected => update coordinate (new coordinate = currentMouseDownPos) => update length line (line has related to point selected) => update area (if area has name => auto delete area name after moving point in area).

case "Point":

        {

          let newLocation = [

            PaintIn.currentMouseDownPos.x,

            PaintIn.currentMouseDownPos.y,

          ];

          obj.point = newLocation;

          obj.x = newLocation[0];

          obj.y = newLocation[1];

          processingData.allLine.forEach((line) => line.getLength());

          processingData.allArea.forEach((area) => {

            for (let line of area.Line) {

              if (

                JSON.stringify(line.Point[0]) === JSON.stringify(obj) ||

                JSON.stringify(line.Point[1]) === JSON.stringify(obj)

              ) {

                area.getPointFlow();

                area.getArea();

                area.getCenter();

                area.getPerimeter();

                area.name = null;

                area.coordNaming = null;

              }

            }

          });

        }

        break;

* ***Line:***

Obj selected

=> Update coordinate for 2 points in line:

+ New coordinate = currentMouseDownPos

+ Caculating the vector moving (the vector has start point is bisecting point of line and end point is currentMouseDownPos)

=> Create 2 new points

=> Determines the point on the selected object that is associated with other points on the difference line

=> Determines these areas have point moved

=> Update the coordinate and the length of all lines whose points are connected to the 2 points of the moved object

=> Update areas have line moved (if areas have name => auto delete area names after moving line in area).

case "Line": {

        //move line

        let point1 = obj.Point[0].point;

        let point2 = obj.Point[1].point;

        let newLocation = [

          PaintIn.currentMouseDownPos.x,

          PaintIn.currentMouseDownPos.y,

        ];

        let centerPoint = math.divide(math.add(point1, point2), 2);

        let translateVect = math.subtract(newLocation, centerPoint);

        let newPoint1 = math.add(point1, translateVect);

        let newPoint2 = math.add(point2, translateVect);

        //create new point obj

        let newPointObj1 = new Point(newPoint1, obj.Point[0].name);

        let newPointObj2 = new Point(newPoint2, obj.Point[1].name);

        let pointLinks1 = [];

        let pointLinks2 = [];

        processingData.allLine.forEach((line) => {

          if (JSON.stringify(line.Point[0]) === JSON.stringify(obj.Point[0])) {

            line.Point[0] = obj.Point[0];

            pointLinks1.push(line.Point[0]);

          } else if (

            JSON.stringify(line.Point[1]) === JSON.stringify(obj.Point[0])

          ) {

            line.Point[1] = obj.Point[0];

            pointLinks1.push(line.Point[1]);

          } else if (

            JSON.stringify(line.Point[0]) === JSON.stringify(obj.Point[1])

          ) {

            line.Point[0] = obj.Point[1];

            pointLinks2.push(line.Point[0]);

          } else if (

            JSON.stringify(line.Point[1]) === JSON.stringify(obj.Point[1])

          ) {

            line.Point[1] = obj.Point[1];

            pointLinks2.push(line.Point[1]);

          }

        });

        let areaChanges = [];

        processingData.allArea.forEach((area) => {

          for (let line of area.Line) {

            let point1 = obj.Point[0];

            let point2 = obj.Point[1];

            if (

              JSON.stringify(line.Point[0]) == JSON.stringify(point1) ||

              JSON.stringify(line.Point[0]) == JSON.stringify(point2) ||

              JSON.stringify(line.Point[1]) == JSON.stringify(point1) ||

              JSON.stringify(line.Point[1]) == JSON.stringify(point2)

            ) {

              processingData.prototype.addObject(area, areaChanges);

            }

          }

        });

        processingData.allLine.forEach((line) => {

          if (pointLinks1.length > 0) {

            for (let point of pointLinks1) {

              if (JSON.stringify(line.Point[0]) === JSON.stringify(point)) {

                line.Point[0] = newPointObj1;

              }

              if (JSON.stringify(line.Point[1]) === JSON.stringify(point)) {

                line.Point[1] = newPointObj1;

              }

            }

          } else {

            obj.Point[0] = newPointObj1;

          }

          if (pointLinks2.length > 0) {

            for (let point of pointLinks2) {

              if (JSON.stringify(line.Point[0]) === JSON.stringify(point)) {

                line.Point[0] = newPointObj2;

              }

              if (JSON.stringify(line.Point[1]) === JSON.stringify(point)) {

                line.Point[1] = newPointObj2;

              }

            }

          } else {

            obj.Point[1] = newPointObj2;

          }

          line.getLength();

        });

        if (areaChanges.length > 0) {

          processingData.allArea.forEach((area) => {

            for (let areaChanged of areaChanges) {

              if (JSON.stringify(area) == JSON.stringify(areaChanged)) {

                area.getPointFlow();

                area.getArea();

                area.getCenter();

                area.getPerimeter();

                area.name = null;

                area.coordNaming = null;

              }

            }

          });

        } else {

          break;

        }

        break;

      }