





















/\* Fonction setRectangle \*/

/\*La fonction setRectangle utilise les valeurs xStart et yStart pour définir le point de départ de la sélection et utilise les valeurs xCourant et yCourant pour définir la position actuelle du curseur. Le ratio est utilisé pour maintenir la proportion des dimensions en cas d'utilisation du mode de proportionnalité.

La fonction utilise des conditions if pour vérifier si le curseur est actuellement

en haut à gauche,

en bas à gauche,

en haut à droite,

ou en bas à droite par rapport au point de départ de la sélection.

Selon la position du curseur, la hauteur et la largeur du rectangle sont redimensionnées en conséquence, ainsi que sa position (top et left). Il utilise également la variable isProportionel pour vérifier si la proportionnalité est activée ou pas. Si la proportionnalité est activée, les dimensions du rectangle sont ajustées en fonction du ratio.

Il est important de noter que cette fonction est destinée à être utilisée avec un élément HTML <div> défini avec l'ID rectangle et qui est utilisé pour afficher le rectangle de sélection. \*/

function setRectangle(xStart, yStart, xCourant, yCourant, ratio) {

// Fonction permettant de redimensionner le rectangle en fonction du déplacement de curseur

if (yStart > yCourant && xStart > xCourant) {

//dessin en haut et à gauche du point de départ

rectangle.style.height = yStart - yCourant + "px";

rectangle.style.top = yCourant + "px";

if (isProportionel === true) {

rectangle.style.width = (yStart - yCourant) \* ratio + "px";

rectangle.style.left = xStart - (yStart - yCourant) \* ratio + "px";

} else {

rectangle.style.width = xStart - xCourant + "px";

rectangle.style.left = xCourant + "px";

}

}

if (yStart < yCourant && xStart > xCourant) {

//dessin en bas et à gauche du point de départ

rectangle.style.height = yCourant - yStart + "px";

rectangle.style.top = yStart + "px";

if (isProportionel === true) {

rectangle.style.width = (yCourant - yStart) \* ratio + "px";

rectangle.style.left = xStart - (yCourant - yStart) \* ratio + "px";

} else {

rectangle.style.width = xStart - xCourant + "px";

rectangle.style.left = xCourant + "px";

}

}

if (yStart > yCourant && xStart < xCourant) {

//dessin en haut et à droite du point de départ

rectangle.style.height = yStart - yCourant + "px";

rectangle.style.top = yCourant + "px";

if (isProportionel === true) {

rectangle.style.width = (yStart - yCourant) \* ratio + "px";

rectangle.style.left = xStart + "px";

} else {

rectangle.style.width = xCourant - xStart + "px";

rectangle.style.left = xStart + "px";

}

}

if (yStart < yCourant && xStart < xCourant) {

//dessin en bas et à droite du point de départ

rectangle.style.height = yCourant - yStart + "px";

rectangle.style.top = yStart + "px";

if (isProportionel === true) {

rectangle.style.width = (yCourant - yStart) \* ratio + "px";

rectangle.style.left = xStart + "px";

} else {

rectangle.style.width = xCourant - xStart + "px";

rectangle.style.left = xStart + "px";

}

}

}

/\* Fonction detourage \*/

function detourage(monImage) {

/\* le but est de récupérer la partie utile de l'image dans un rectangle \*/

var tmpCanvas = document.createElement("canvas"); //creation d'un canvas

tmpCanvas.width = monImage.naturalWidth; //définition de la largeur

tmpCanvas.height = monImage.naturalHeight; //définition de la hauteur

var tmpContext = tmpCanvas.getContext("2d"); //récupération du contexte

tmpContext.drawImage(monImage, 0, 0); //dessin de l'image

var tmpMap = tmpContext.getImageData(0, 0, tmpCanvas.width, tmpCanvas.height); //récupération du Image Data

var tmpImgData = tmpMap.data; //création d'un raccourci vers les datas

var whiteLine;

var whiteCol = true;

var nbLigne = 0;

var nbColonne = 0;

var i,j,k;

var meslignes = [];

var mescolonnes = [];

for ( i = 0; i < tmpImgData.length; i += monImage.naturalWidth \* 4) {

// cette boucle permet de parcourir chaque ligne de pixels de l'image

nbLigne++;

whiteLine = true;

for (j = 0; j < (monImage.naturalWidth \* 4); j += 4) {

k = i + j;

if (tmpImgData[k] <= transValue) { whiteLine =false;}

if (tmpImgData[k + 1] <= transValue) {whiteLine =false;}

if (tmpImgData[k + 2] <= transValue) {whiteLine =false;}

}

if (!whiteLine) {meslignes.push(nbLigne);}

}

for (i = 0; i < monImage.naturalWidth \* 4; i += 4) {

// cette boucle permet de parcourir chaque colonne de pixels de l'image

nbColonne++

whiteCol = true;

for (j = 0; j < tmpImgData.length; j += monImage.naturalWidth \* 4) {

k = i+j;

if (tmpImgData[k] <= transValue) { whiteCol =false;}

if (tmpImgData[k + 1] <= transValue) {whiteCol =false;}

if (tmpImgData[k + 2] <= transValue) {whiteCol =false;}

}

if (!whiteCol) {mescolonnes.push(nbColonne);}

}

canvasT.width = mescolonnes.length ;

canvasT.height = meslignes.length ;

tmpMap = tmpContext.getImageData(mescolonnes[0] -1, meslignes[0] -1, mescolonnes.length + 20, meslignes.length + 20);

ctxT.putImageData(tmpMap, 0, 0);

var image = new Image();

image.src = canvasT.toDataURL();

return image;

}

/\* Fonction drawImageInPerspective \*/

/\* Cette fonction permet de dessiner une image (donnée par l'argument srcImg) sur un canvas (donné par l'argument targetCanvas) en utilisant une transformation de perspective pour adapter l'image aux coordonnées des quatres coins (topLeftX, topLeftY, bottomLeftX, bottomLeftY, topRightX, topRightY, bottomRightX, bottomRightY) du quadrilatère souhaitée sur le canvas.

Elle commence par définir la taille de l'image d'origine en utilisant les propriétés naturalWidth et naturalHeight de l'image. Puis elle calcule les dimensions du quadrilatère cible en utilisant les coordonnées des quatres coins.

Par la suite, elle crée un nouveau canvas temporaire, et utilise un contexte 2D pour y dessiner l'image d'origine en utilisant les transformations de translation, de miroir (si flipHorizontally ou flipVertically est vrai) et de mise à l'échelle pour adapter l'image aux dimensions du quadrilatère cible.

Enfin, elle utilise la méthode getImageData pour obtenir un tableau unidimensionnel contenant les données des pixels de l'image temporaire, et parcoure ce tableau pour mettre à jour les pixels correspondants sur le canvas cible. \*/

function drawImageInPerspective(

srcImg,

targetCanvas,

//Définir où sur le canva l'image doit être dessinée :

//coordonnées des 4 coins du quadrilatère sur lesquels l'image rectangulaire d'origine sera transformée :

topLeftX,

topLeftY,

bottomLeftX,

bottomLeftY,

topRightX,

topRightY,

bottomRightX,

bottomRightY,

//Facultativement retourner l'image d'origine horizontalement ou verticalement \*avant\* de transformer l'image rectangulaire d'origine en quadrilatère convexe :

flipHorizontally,

flipVertically,

transparence = false

) {

var srcWidth = srcImg.naturalWidth;

var srcHeight = srcImg.naturalHeight;

var targetMarginX = Math.min(topLeftX, bottomLeftX, topRightX, bottomRightX);

var targetMarginY = Math.min(topLeftY, bottomLeftY, topRightY, bottomRightY);

var targetTopWidth = topRightX - topLeftX;

var targetTopOffset = topLeftX - targetMarginX;

var targetBottomWidth = bottomRightX - bottomLeftX;

var targetBottomOffset = bottomLeftX - targetMarginX;

var targetLeftHeight = bottomLeftY - topLeftY;

var targetLeftOffset = topLeftY - targetMarginY;

var targetRightHeight = bottomRightY - topRightY;

var targetRightOffset = topRightY - targetMarginY;

var tmpWidth = Math.max(

targetTopWidth + targetTopOffset,

targetBottomWidth + targetBottomOffset

);

var tmpHeight = Math.max(

targetLeftHeight + targetLeftOffset,

targetRightHeight + targetRightOffset

);

var tmpCanvas = document.createElement("canvas");

tmpCanvas.width = tmpWidth;

tmpCanvas.height = tmpHeight;

var tmpContext = tmpCanvas.getContext("2d");

tmpContext.translate(

flipHorizontally ? tmpWidth : 0,

flipVertically ? tmpHeight : 0

);

tmpContext.scale(

(flipHorizontally ? -1 : 1) \* (tmpWidth / srcWidth),

(flipVertically ? -1 : 1) \* (tmpHeight / srcHeight)

);

tmpContext.drawImage(srcImg, 0, 0);

var tmpMap = tmpContext.getImageData(0, 0, tmpWidth, tmpHeight);

var tmpImgData = tmpMap.data;

var targetContext = targetCanvas.getContext("2d");

var targetMap = targetContext.getImageData(

targetMarginX,

targetMarginY,

tmpWidth,

tmpHeight

);

var targetImgData = targetMap.data;

var targetX, targetY, tmpPoint, targetPoint, tmpY, tmpX;

// Passer le rectangle contenant notre image en perspective en négatif, utilisation pour test

/\*

for (var i = 0; i < targetImgData.length; i += 4) {

targetImgData[i] = 255 - targetImgData[i];

targetImgData[i + 1] = 255 - targetImgData[i + 1];

targetImgData[i + 2] = 255 - targetImgData[i + 2];

}

\*/

for (var tmpY = 0; tmpY < tmpHeight; tmpY++) {

for (var tmpX = 0; tmpX < tmpWidth; tmpX++) {

tmpPoint = (tmpY \* tmpWidth + tmpX) \* 4;

targetX =

(targetTopOffset + (targetTopWidth \* tmpX) / tmpWidth) \*

(1 - tmpY / tmpHeight) +

(targetBottomOffset + (targetBottomWidth \* tmpX) / tmpWidth) \*

(tmpY / tmpHeight);

targetX = Math.round(targetX);

//Ici la coordonnée Y du point d'origine est transposée en coordonnée cible (simulacre de perspective):

targetY =

(targetLeftOffset + (targetLeftHeight \* tmpY) / tmpHeight) \*

(1 - tmpX / tmpWidth) +

(targetRightOffset + (targetRightHeight \* tmpY) / tmpHeight) \*

(tmpX / tmpWidth);

targetY = Math.round(targetY);

targetPoint = (targetY \* tmpWidth + targetX) \* 4;

// Décalage de pixel pour couvrir les pixels morts

if (!transparence) {

if (targetPoint >= 4 && targetPoint < targetImgData.length - 5) {

if (targetX != 0) {

targetImgData[targetPoint - 4] = tmpImgData[tmpPoint]; //rouge

targetImgData[targetPoint + 1 - 4] = tmpImgData[tmpPoint + 1]; //vert

targetImgData[targetPoint + 2 - 4] = tmpImgData[tmpPoint + 2]; //bleu

targetImgData[targetPoint + 3 - 4] = tmpImgData[tmpPoint + 3]; //alpha

}

}

// Pixels cibles

targetImgData[targetPoint] = tmpImgData[tmpPoint]; //rouge

targetImgData[targetPoint + 1] = tmpImgData[tmpPoint + 1]; //vert

targetImgData[targetPoint + 2] = tmpImgData[tmpPoint + 2]; //bleu

targetImgData[targetPoint + 3] = tmpImgData[tmpPoint + 3]; //alpha

} else {

if (tmpImgData[tmpPoint] < transValue) {

if (tmpImgData[tmpPoint + 1] < transValue) {

if (tmpImgData[tmpPoint + 2] < transValue) {

if (targetPoint >= 4 && targetPoint < targetImgData.length - 5) {

if (targetX != 0) {

targetImgData[targetPoint - 4] = tmpImgData[tmpPoint];

targetImgData[targetPoint + 1 - 4] = tmpImgData[tmpPoint + 1];

targetImgData[targetPoint + 2 - 4] = tmpImgData[tmpPoint + 2];

targetImgData[targetPoint + 3 - 4] = tmpImgData[tmpPoint + 3];

}

}

// Pixels cibles

targetImgData[targetPoint] = tmpImgData[tmpPoint]; //rouge

targetImgData[targetPoint + 1] = tmpImgData[tmpPoint + 1]; //vert

targetImgData[targetPoint + 2] = tmpImgData[tmpPoint + 2]; //bleu

targetImgData[targetPoint + 3] = tmpImgData[tmpPoint + 3]; //alpha

}

}

}

}

}

}

targetContext.putImageData(targetMap, targetMarginX, targetMarginY);

ctx1.drawImage(canvas0, 0, 0);

}

/\* Fonction setQuadPerspective \*/

/\*La fonction "setQuadPerspective" vérifie si un quadrilatère est convexe en utilisant une méthode par force brute : en calculant les produits croisés des vecteurs formés par ses côtes pour chaque combinaison de quatre points d'un tableau "monQuad". Elle verifie egalement que les quatre points du quadrilatère sont distincts et que le point en haut à gauche a une valeur y plus élevée que le point en bas à gauche, que le point en haut à droite a une valeur y plus élevée que le point en bas à droite, que le point en haut à gauche a une valeur x plus basse que le point en haut à droite et que le point en bas à gauche a une valeur x plus basse que le point en bas à droite. Elle crée plusieurs boucles et vérifie un grand nombre de combinaisons.\*/

function setQuadPerspective() {

let i, j, k, l;

let topLeftX = -1,

topLeftY = -1,

bottomLeftX = -1,

bottomLeftY = -1,

topRightX = -1,

topRightY = -1,

bottomRightX = -1,

bottomRightY = -1;

compoint = 0; //On vérifie que chaque point est bien distinct

/\*Pour vérifier que les diagonales d'un quadrilatère ABCD se coupent en son sein ( === quadrilatère convexe) on compare les deux valeurs

[(xC-xA)(yB-yA)-(yC-yA)(xB-xA)]\*[(xD-xA)(yB-yA)-(yD-yA)(xB-xA)] et

[(xA-xC)(yD-yC)-(yA-yC)(xD-xC)]\*[(xB-xC)(yD-yC)-(yB-yC)(xD-xC)]

\*/

for (i = 0; i < 7; i = i + 2) {

for (j = 0; j < 7; j = j + 2) {

if (j != i) {

for (k = 0; k < 7; k = k + 2) {

if (k != j && k != i) {

for (l = 0; l < 7; l = l + 2) {

if (l != i && l != j) {

if (l != k) {

var cond1 =

((monQuad[j] - monQuad[i]) \*

(monQuad[k + 1] - monQuad[i + 1]) -

(monQuad[j + 1] - monQuad[i + 1]) \*

(monQuad[k] - monQuad[i])) \*

((monQuad[l] - monQuad[i]) \*

(monQuad[k + 1] - monQuad[i + 1]) -

(monQuad[l + 1] - monQuad[i + 1]) \*

(monQuad[k] - monQuad[i]));

var cond2 =

((monQuad[i] - monQuad[j]) \*

(monQuad[l + 1] - monQuad[j + 1]) -

(monQuad[i + 1] - monQuad[j + 1]) \*

(monQuad[l] - monQuad[j])) \*

((monQuad[k] - monQuad[j]) \*

(monQuad[l + 1] - monQuad[j + 1]) -

(monQuad[k + 1] - monQuad[j + 1]) \*

(monQuad[l] - monQuad[j]));

if (cond1 < 0 && cond2 < 0) {

/\*En prenant TL(TopLeft), TR(TopRight), BL(BottomLeft) et BR(BottomRight)

Voici les conditions à remplir pour dessiner un quadrilatère sur le Canvas

-le point TL doit être plus haut que le point BL

-le point TR doit être plus haut que le point BR

-le point TL doit être à gauche du point TR

-le point BL doit être à gauche du point BR\*/

if (

monQuad[i + 1] < monQuad[l + 1] &&

//le point TL doit être plus haut que le point BL

monQuad[j + 1] < monQuad[k + 1]

//le point TR doit être plus haut que le point BR

) {

if (

monQuad[i] < monQuad[j] &&

//le point TL doit être à gauche du point TR

monQuad[l] < monQuad[k]

//le point BL doit être à gauche du point BR

) {

if (

monQuad[i + 1] < monQuad[k + 1] &&

//le point TL doit être à gauche du point TR

monQuad[j + 1] < monQuad[l + 1]

//le point BL doit être à gauche du point BR

) {

/\* Assignation des valeurs triées\*/

bottomRightX = monQuad[k];

bottomRightY = monQuad[k + 1];

topRightX = monQuad[j];

topRightY = monQuad[j + 1];

bottomLeftX = monQuad[l];

bottomLeftY = monQuad[l + 1];

topLeftX = monQuad[i];

topLeftY = monQuad[i + 1];

}

}

}

}

}

}

}

}

}

}

}

}

//Purge et réaffectation du conteneur monQuad

monQuad = [];

monQuad.push(topLeftX);

monQuad.push(topLeftY);

monQuad.push(topRightX);

monQuad.push(topRightY);

monQuad.push(bottomRightX);

monQuad.push(bottomRightY);

monQuad.push(bottomLeftX);

monQuad.push(bottomLeftY);

for (var xy in monQuad) {

if (monQuad[xy] === -1) {

return false;

}

}

return true;

}