/\* 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);

}