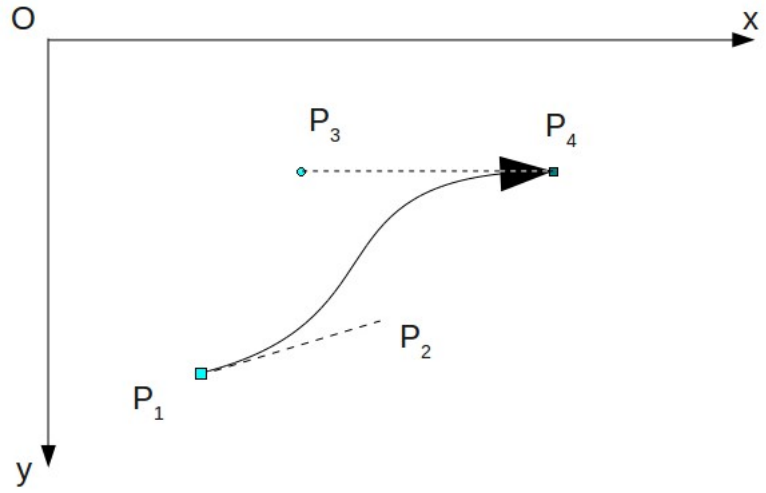
Algorithme de création de flèches Auteur : Eric Bachard décembre 2017

Thème général : création d'un canvas dans un logiciel écrit en C++

But de ce document : proposer un algorithme pour calculer les coordonnées des points permettant de dessiner une flèche dont la largeur et la longueur seront contrôlables, cette flèche étant placée à l'extrémité d'une ligne ou d'une courbe (type courbe de Bezier par exemple).

Outils mathématiques pré-requis : le produit scalaire entre 2 vecteurs, et l'exploitation de l'orthogonalité de 2 vecteurs à l'aide du produit scalaire.

cf figure ci-dessous (réalisée avec OOoLight, un fork d'OpenOffice.org dont je suis le développeur principal)



La courbe est une courbe de Bezier, mais l'étude simplifiée se fera avec une simple droite, car le principe est le même.

Repère utilisé, et conventions de notations :

Le repère Oxy sera orienté cf la figure ci-dessous, c'est à dire avec l'axe Oy **orienté vers le bas** (comme par exemple celui utilisé avec OpenGL), et on notera x_A et y_A les composantes du point A dans le repère Oxy.

Principe du calcul des coordonnées

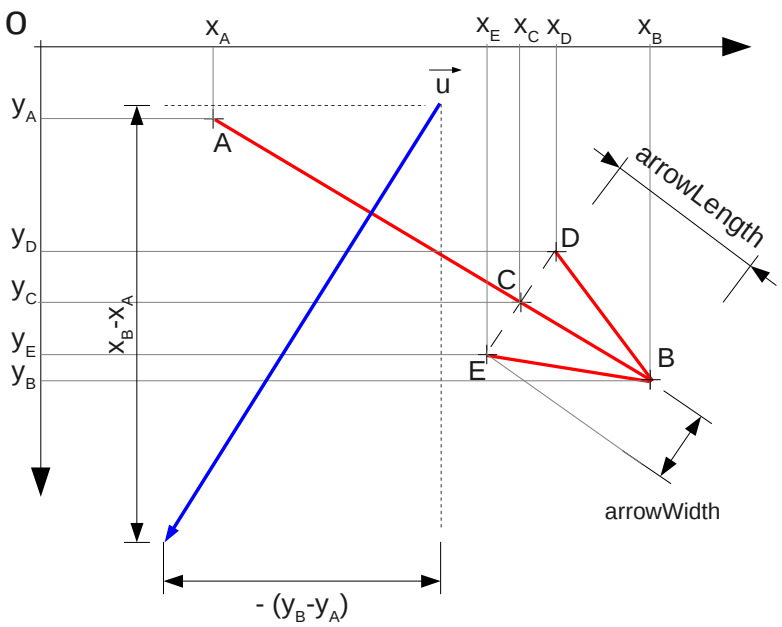
L'essentiel du problème va consister à calculer les coordonnées des points C, D et E.

Pour les calculs, on utilisera les points A, B, C, D et E comme décrits ci-dessous.

Pour les notations, AB désignera la longueur de la courbe, CB la longueur de la flèche (notée arrowLength), la largeur de la flèche (que ce soit la distance DC ou la distance CE) sera notée arrowWidth

Mise en équation

Pour plus de clarté, on va dessiner la flèche penchée vers le bas, vers la droite.



Coordonnées du point C

Compte tenu de l'orientation de la flèche par rapport au repère, et en appliquant le théorème de Thalès dans le triangle formé par les droites x_A x_B et AB il vient :

$$\begin{cases} \frac{x_B - x_C}{x_B - x_A} = \frac{arrowLength}{AB} \\ \frac{y_B - y_C}{y_B - y_A} = \frac{arrowLength}{AB} \end{cases} \Rightarrow \begin{cases} x_B - x_C = \frac{arrowLength.(x_B - x_A)}{AB} \\ y_B - y_C = \frac{arrowLength.(y_B - y_A)}{AB} \end{cases}$$

$$\Rightarrow \begin{cases} x_C = x_B - \frac{arrowLength.(x_B - x_A)}{AB} \\ y_C = y_B - \frac{arrowLength.(y_B - y_A)}{AB} \end{cases}$$ ou encore $$\boxed{\begin{cases} \boldsymbol{x_C = x_B + \frac{arrowLength.(x_A - x_B)}{AB}} \\ \boldsymbol{y_C = y_B + \frac{arrowLength.(y_A - y_B)}{AB}} \end{cases}}$$

Coordonnées du points D

Comme illustré sur le dessin ci-dessus, les vecteurs $\overrightarrow{DC}$, et $\overrightarrow{AB}$ dont les composantes (respectivement) $\begin{cases} x_C - x_D (< 0 \text{ sur le dessin}) \\ y_C - y_D \end{cases}$ et $\begin{cases} x_B - x_A \\ y_B - y_A \end{cases}$ sont perpendiculaires. Soit maintenant $\vec{u}$ un vecteur de longueur AB, et dirigé de droite à gauche sur le dessin, et perpendiculaire à $\overrightarrow{AB}$. Ce vecteur vérifiera que $\vec{u}.\overrightarrow{AB}=0$, et ses composantes seront telles que $\|\vec{u}\|=\|\overrightarrow{AB}\|$. On vérifie aisément sur le dessin que les composantes de $\vec{u}$ $\begin{cases} -(y_B - y_A) \\ x_B - x_A \end{cases}$ vérifient bien $\vec{u}.\overrightarrow{AB}=0 \Leftrightarrow \begin{pmatrix} -(y_B - y_A) \\ x_B - x_A \end{pmatrix}.\begin{pmatrix} (x_B - x_A) \\ y_B - y_A \end{pmatrix}=0$. Enfin, comme $\overrightarrow{DC}$ est orienté comme $\frac{\vec{u}}{AB}$, et que $\overrightarrow{DC}$ a pour longueur arrowWidth, il vient : $\boldsymbol{\overrightarrow{DC} = arrowWidth.\frac{\vec{u}}{AB}}$.

Ceci entraîne: $\begin{cases} x_C - x_D = -\frac{arrowWidth.(y_B - y_A)}{AB} \\ y_C - y_D = +\frac{arrowWidth(x_B - x_A)}{AB} \end{cases}$ => $\begin{cases} x_C = x_D - \frac{arrowWidth.(y_B - y_A)}{AB} \\ y_C = y_D + \frac{arrowWidth(x_B - x_A)}{AB} \end{cases}$ =>

$$\boxed{\begin{cases} \boldsymbol{x_D = x_C + \frac{arrowWidth.(y_B - y_A)}{AB}} \\ \boldsymbol{y_D = y_D - \frac{arrowWidth(x_B - x_A)}{AB}} \end{cases}}$$ (/!\ D dépend de C)

Coordonnées du point E

Comme pour $\overrightarrow{DC}$, on sait aussi que $\overrightarrow{DC}$ est orienté comme $\frac{\vec{u}}{AB}$ (donc colinéaire à $\vec{u}$) et que ce vecteur a pour longueur arrowWidth. Pour le point E, cela permet d'écrire : $\boldsymbol{\overrightarrow{CE} = arrowWidth.\frac{\vec{u}}{AB}}$

Ceci entraîne: $\begin{cases} x_E - x_C = -\frac{arrowWidth.(y_B - y_A)}{AB} \\ y_E - y_C = +\frac{arrowWidth(x_B - x_A)}{AB} \end{cases}$ => $\begin{cases} x_E = x_C - \frac{arrowWidth.(y_B - y_A)}{AB} \\ y_E = y_C + \frac{arrowWidth(x_B - x_A)}{AB} \end{cases}$

=>
$$\boxed{\begin{cases} \boldsymbol{x_E = x_C - \frac{arrowWidth.(y_B - y_A)}{AB}} \\ \boldsymbol{y_E = y_C + \frac{arrowWidth(x_B - x_A)}{AB}} \end{cases}}$$ (/!\ E dépend aussi de C)

Algorithme de création de flèches Auteur : Eric Bachard décembre 2017

Algorithme utilisé pour dessiner la flèche

On suppose que les points appartiennent à un tableau (ou un vecteur) de points, appelé points, de longueur paire (chaque flèche dessinée = 2 points).
On supposera de plus que si la longueur AB est inférieure à 1,5 fois la longueur de la flèche, on ne dessinera pas la flèche (ce choix est purement arbitraire, et surtout esthétique)
Pour chaque flèche, le point A, sera symbolisé par le point noté points[i], et le point B par le point noté points[i+1].

Sous programme calcul_flèches

Variables utilisées :
Entier **numéro_point, nombre_elements** (pair)
Tableau de points **P[indice]**, de dimension 1 et de taille nombre_elements (forcément pair)
Points **A** (réel x_A , réel y_A), **B**, **C** et D
Réel AB

```
Pour numéro de point  allant de 1 à nombre_elements, par pas de 2
   FAIRE
     Calculer la distance AB entre le point numero_point et le point d'indice numero_point + 1
     SI (longueur AB > 1,5 fois la longueur de la flèche)
        FAIRE :
           pointC(  points[i+1].x - (arrowLength * (points[i+1].x - points[i].x))/AB,
                    points[i+1].y - (arrowLength * (points[i+1].y - points[i].y))/AB);
            pointD(  pointC.x + (arrowWidth*(points[i+1].y -points[i].y))/AB,
                    pointC.y - (arrowWidth*(points[i+1].x -points[i].x))/AB);
            pointE(  pointC.x - (arrowWidth*(points[i+1].y -points[i].y))/AB,
                    pointC.y + (arrowWidth*(points[i+1].x -points[i].x))/AB);
        FIN SI
FIN POUR
```

Exemple d'implémentation

Dans l'exemple qui suit, écrit en C++, on utilise le mode de création dit Mode Immédiat de l'interface utilisateur, et le calcul des coordonnées des points est immédiatement suivi du stockage des valeurs dans la pile d'objets qui seront dessinés à l'appel de ImGui::Render().

```
for (int i = 0; i < points.Size-1; i += 2)
   {
      if (draw_arrows == true)
      {
         // ARROW + P1 (point A), P2 (control-point1), P3(control-point2), P4) -> courbe de Bezier
         P1P4 = sqrtf( (   points[i+1].x -points[i].x)*(points[i+1].x -points[i].x)
                         + (points[i+1].y  -points[i].y)*(points[i+1].y - points[i].y) );
         if (P1P4 > 1.5f * arrowLength)
         {
            ImVec2 pointC(  points[i+1].x - (arrowLength * (points[i+1].x - points[i].x))/P1P4,
                            points[i+1].y - (arrowLength * (points[i+1].y - points[i].y))/P1P4);
            ImVec2 pointD(  pointC.x + (arrowWidth*(points[i+1].y -points[i].y))/P1P4,
                            pointC.y - (arrowWidth*(points[i+1].x -points[i].x))/P1P4);
           ImVec2 pointE(  pointC.x - (arrowWidth*(points[i+1].y -points[i].y))/P1P4,
                            pointC.y + (arrowWidth*(points[i+1].x -points[i].x))/P1P4);
          float offsetX = canvas_pos.x;
            float offsetY = canvas_pos.y;
            float thickness = 1.5f;
            draw_list->PathClear();
            draw_list->PathLineTo(ImVec2(pointD.x + offsetX, pointD.y + offsetY));
            draw_list->PathLineTo(ImVec2(points[i+1].x + offsetX, points[i+1].y + offsetY));
            draw_list->PathLineTo(ImVec2(pointE.x + offsetX, pointE.y + offsetY));
            draw_list->PathStroke(colors[i], false, thickness);

            if ((show_shooters) && (i < shooters.Size))
               draw_list->AddText(ImGui::GetFont(), ImGui::GetFontSize(),
               ImVec2( canvas_pos.x+points[i].x - 2.0f ,
                           canvas_pos.y+points[i].y+5.0f),
                           ImColor(0,0,0,255),
                           shooters[i], NULL, 0.0f, &clip_rect);
      }
}
```

La réalisation de ce document m'a été inspirée par le travail de M. Yoann Morel, que vous pourrez retrouver en allant visiter le lien ci-dessous :
http://xymaths.free.fr/Informatique-Programmation/javascript/canvas-dessin-fleche.php