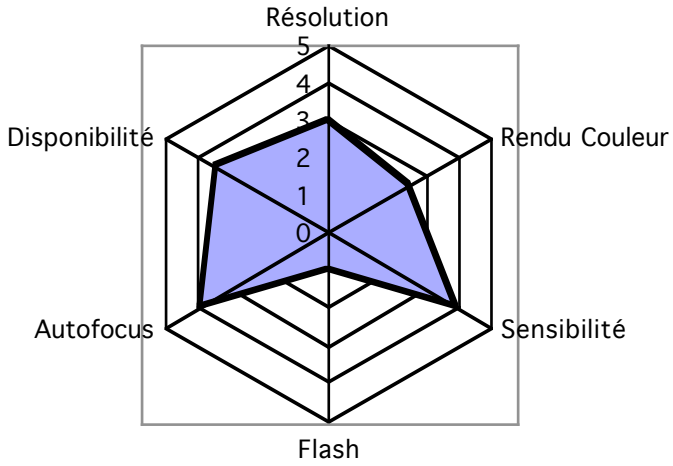
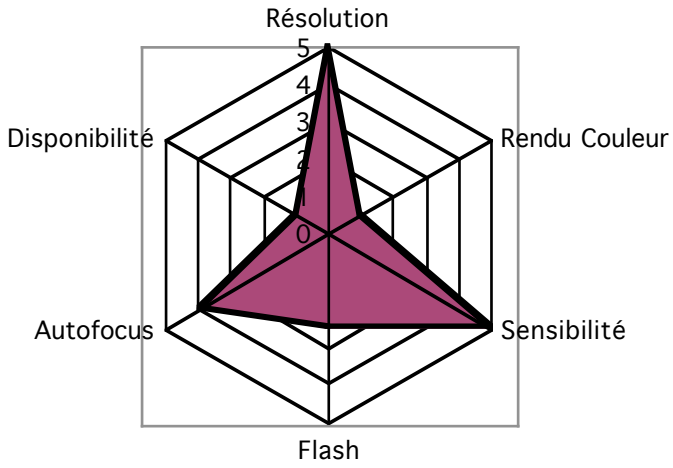
Définition

Les graphes de Kiviat (ou graphes radars suivant la terminologie utilisée par certains logiciels grand public) permettent de représenter des données variées relatives à un objet sur un graphe unique et de faciliter la comparaison entre deux objets.

Voici un exemple de graphe de Kiviat demandé par notre client, vendeur d'appareil photo :



Et voici un deuxième graphe pour un autre appareil :



La comparaison des deux graphes est très aisé et si vous multipliez le nombre de graphes à comparer vous allez pouvoir faire ressortir rapidement des tendances de formes : vous aurez les appareils bien équilibré qui auront un graphe plutôt « rond » alors que les appareils spécialisés dans un domaine présenteront une forme en pointe ou en losange écrasé.

Tracer un graphe de Kiviat

Comment dessiner de tel graphe dans une zone 4D Chart ?

La lecture de la documentation nous confirme que ces graphes n'existent pas en standard, et de plus il n'est pas possible de dessiner deux graphes dans une même zone sans passer par la programmation.

Par contre nous avons remarquer que 4D Chart propose des outils permettant de dessiner par programmation des lignes, des cercles, des rectangles, des polygones … Nous sentons donc que nous pourrons nous en sortir en utilisant en outre quelques fonctions mathématiques proposées par le langage de 4D.

Afin de dessiner le graphe nous supposons que nous avons un formulaire avec une zone 4D Chart d'une taille de 600 x 600 pixels et nommée « zChart ». Un bouton ayant le label « Dessin » permet de déclencher le code que nous allons expliquer ci-dessous.

Pour commencer notre graphe nous allons initialiser la zone 4D Chart.

```
ch_EXECUTER MENU (zChart;1001)
```

Le code 1001 correspond à l'item de menu nouveau du menu fichier. Nous commençons donc par rendre vierge la zone de travail.

Nous allons définir à 200 pixels des bords de la zone 4D Chart le centre du graphe grâce à deux variables :

```
zCentre_x:=300

zCentre_y:=300
```

De même nous définisons le rayon du graphe à 150 pixels :

```
zRayon:=150
```

Pour dessiner un graphe, il faut avoir des valeurs ! Nous avons donc un tableau de réel contenant les différentes valeurs du graphe :

```
TABLEAU REEL(_valeurs;6)

_valeurs{1}:=3
_valeurs{2}:=2,5
_valeurs{3}:=4
_valeurs{4}:=1
_valeurs{5}:=4
```

```
_valeurs{6}:=3,5
```

Nous définissons de même pour définir les légendes des séries :

```
TABLEAU ALPHA(20;_legende_serie;6)

_legende_serie{1}:="Résolution"
_legende_serie{2}:="Rendu couleur"
_legende_serie{3}:="Sensibilité"
_legende_serie{4}:="Flash"
_legende_serie{5}:="Autofocus"

_legende_serie{6}:="Sensibilité"
```

Nous pouvons en déduire le nombre d'axes du graphe de Kiviat , et la valeur maximum a afficher :

```
zNbValeurs:=Taille tableau(_valeurs)

$Max:=0
Boucle ($i;1;zNbValeurs;1)
 Si ($Max<_valeurs{$i})
  $Max:=_valeurs{$i}
 Fin de si
Fin de boucle

$Max:=Ent($Max)+1
```

Nous pouvons à présent commencer le dessin proprement dit. Nous dessinerons dans l'ordre afin de respecter les effets de superposition :

- Les lignes de niveau du fond (formant une sorte de toile d'araignée)
- Le polygone des valeurs
- Les axes
- L'échelle
- Les noms des axes

Le dessin des lignes de niveau utilise la fonction ch_Tableau vers polygone de 4D Chart permettant de dessiner un polygone défini par ses sommets.

Chaque valeur sera affichée sur un axe incliné d'un certain angle par rapport à l'horizontale. Cet angle dépend du nombre d'axes à dessiner. Cette série d'angles va servir un certain nombre de fois dans les calculs, il nous semble donc judicieux de les stocker temporairement dans un tableau dédié. Ceci présente l'avantage de réduire le code et de simplifier la maintenance si nous désirons faire évoluer le calcul des angles.
Ici se pose le problème du sens horaire et du sens trigonométrique pour le calcul des angles.
Dans 4D, les fonctions trigonométriques utilisent des angles définis en radians ; elles sont basées sur le cercle trigonométrique qui « tourne » dans le sens anti-horaire. Par contre les habitudes pour le dessin des graphes de Kiviat (comme pour les diagrammes en secteurs) veulent que l'on respecte le sens de lecture usuel c'est-à-dire en tournant dans le sens horaire. Nous devons donc en tenir compte pour calculer notre série d'angles

ordonnée. De plus la lecture commence en haut du cercle (à 12 heures) alors que le cercle trigonométrique commence à droite (à 3 heures). Nous devons donc décaler d'autant nos angles.
Dans le code, nous utilisons la constante Degré qui permet de convertir facilement une valeur en degré vers une valeur en radians.
Voici le calcul des angles du graphe.

```
TABLEAU REEL($_angle;zNbValeurs)

$sens:=-1
$angle_depart:= 90*Degré
Boucle ($Point;1;zNbValeurs)
 $_angle{$Point}:=$angle_depart+(360*Degré /zNbValeurs*($Point-1)*$sens)

Fin de boucle
```

La fonction ch_Tableau vers polygone nécessite deux tableaux de valeurs ; les abscisses et les ordonnées :

```
TABLEAU REEL($_x;zNbValeurs+1)

TABLEAU REEL($_y;zNbValeurs+1)
```

Vous remarquerez qu'il y a une valeur de plus dans le tableau que le nombre de valeurs à afficher. Ceci est nécessaire pour fermer le polygone, en donnant à la fonction comme dernier point un point ayant les mêmes valeurs pour son abscisse et son ordonnée que le premier point.

Nous allons commencer par tracer les lignes de valeur. Il nous faut remplir les deux tableaux. Dans cet exemple, nous savons que les valeurs sont comprises entre 1 et 5. Nous fixons donc le pas entre les lignes à 1. Dans un autre cas de figure, il faudra calculer la valeur de ce pas.

```
$pas:=1
```

Nous allons imbriquer deux boucles ; une pour calculer les différentes lignes et une autre imbriquée pour calculer les points de chaque ligne.
L'échelle étant linéaire, nous calculons le « rayon » de la ligne par une simple règle de trois.
Pour calculer les abscisses et les ordonnées, nous nous trouvons face à un nouveau problème. En effet le repère orthonormé classique utilise un axe horizontal numéroté de gauche à droite et un axe vertical numéroté de bas en haut. Malheureusement 4D Chart s'appuie sur les axes des écrans informatiques basés sur un axe horizontal numéroté de gauche à droite et un axe vertical numéroté de haut en bas.
Nous avons donc besoin de procéder à un changement de repère. De plus nous devons décaler l'origine afin que le centre du graphe corresponde au croisement des axes :

Xécran = Xcentre + Xtrigo
Yécran = Ycentre - Ytrigo

Reste à fixer le tramage de manière à ce que le polygone soit non remplis.

```
Boucle ($ligne;1;$Max;$pas)
 $rayon_ligne:=zRayon*$ligne/$Max
 Boucle ($Point;1;zNbValeurs)
  $_x{$Point}:=zCentre_x+(Cos($_angle{$Point})*$rayon_ligne)
  $_y{$Point}:=zCentre_y-(Sin($_angle{$Point})*$rayon_ligne)
 Fin de boucle
 $_x{zNbValeurs+1}:=$_x{1}
 $_y{zNbValeurs+1}:=$_y{1}
  $poly:=ch_Tableau vers polygone (zChart;$_x;$_y)
 ch_FIXER TRAMAGE (zChart;$poly;1;-1)
Fin de boucle
```

Suivant des principes similaires nous pouvons tracer le polygone des valeurs. Ici la position de la valeur sur l'axe doit être calculé pour chaque valeur ! Nous utilisons également une règle de trois par rapport au rayon maximum et à la valeur maximum. Le tramage cette fois sera « plein » (valeur 3) et aura une couleur rouge (valeur 4).

```
Boucle ($Point;1;zNbValeurs)
 $_x{$Point}:=zCentre_x+(Cos($_angle{$Point})*zRayon*_valeurs{$Point}/$Max)
 $_y{$Point}:=zCentre_y-(Sin($_angle{$Point})*zRayon*_valeurs{$Point}/$Max)
Fin de boucle
$_x{zNbValeurs+1}:=$_x{1}
$_y{zNbValeurs+1}:=$_y{1}
$polygone:=ch_Tableau vers polygone (zChart;$_x;$_y)
ch_FIXER TRAMAGE (zChart;$polygone;3;ch_Index vers couleur (4))
```

Le dessin des axes utilise la commande ch_Creer trait. Nous traçons les traits depuis le centre du cercle jusqu'à la ligne extérieure suivant les angles de notre tableau.

```
Boucle ($Point;1;zNbValeurs)
$ligne:=ch_Creer trait (zChart; zCentre_x; zCentre_y;
   zCentre_x+(Cos($_angle{$Point})*zRayon); zCentre_x-(Sin($_angle{$Point})*zRayon))
Fin de boucle
```

Pour le dessin des valeurs de l'axe principal (l'axe correspondant au premier angle du tableau) nous utilisons la commande ch_Créer texte. La position du texte est calculée pour que le chiffre s'affiche à gauche de l'axe (si l'axe principal est vertical). La commande ch_FIXER TEXTE permet de spécifier les attributs du texte. Ici la valeur 2 correspond à la justification à droite.

```
Boucle ($ligne;1;$Max;$pas)
 $Point:=1
  $rayon_ligne:=zRayon*$ligne/$Max
```

```
 $x:=zCentre_x+(Cos($_angle{$Point})*$rayon_ligne)
 $y:=zCentre_y-(Sin($_angle{$Point})*$rayon_ligne)
 $texte:=ch_Creer texte (zChart;$x-20;$y+5;$x-2;$y+15;Chaine($ligne))
ch_FIXER TEXTE (zChart;$texte;ch_Numero de police ("Geneva");9;Gras ; ¬
ch_Index vers couleur (16);2)

Fin de boucle
```

Reste pour que notre graphe soit complet à dessiner les légende en bout des axes. Ici se pose le problème de la position du texte à l'extérieur du graphe. Il faut donc situer correctement les blocs de texte ; c'est le rôle du code contenu dans les deux structures de contrôle Si …sinon … fin de si. $justif :=0 et $justif :=2 signifie que la justification est respectivement à gauche ou à droite. La couleur 16 est le noir.

```
Boucle ($Point;1;zNbValeurs)

 $x:=zCentre_x+(Cos($_angle{$Point})*zRayon)
 $y:=zCentre_x-(Sin($_angle{$Point})*zRayon)
Si ($x>=zCentre_x)
  $x1:=$x+10
  $x2:=$x+100
  $justif:=0
Sinon
  $x1:=$x-10
  $x2:=$x-100
  $justif:=2
Fin de si
Si ($y>=zCentre_y)
  $y1:=$y+5
  $y2:=$y+15
Sinon
  $y1:=$y-5
  $y2:=$y-15
Fin de si
 $texte:=ch_Creer texte (zChart;$x1;$y1;$x2;$y2;_legende_serie{$Point})
ch_FIXER TEXTE (zChart;$texte; ch_Numero de police ("Geneva"); 9; Normal ; ¬
ch_Index vers couleur (16); $justif)

Fin de boucle
```

Rendre générique

Maintenant que nous savons dessiner un graphe de toute pièce, il est temps de passer à la seconde partie de cette note technique en proposant des principes pour rendre générique le dessin d'un graphe de Kiviat.

En étudiant de près le code proposé précédemment nous pouvons découper en plusieurs parties logiques :

- la définition des valeurs à dessiner,
- le choix des paramètres utilisés pour tracer le graphe,
- le dessin du graphe proprement dit.

Nos méthodes génériques vont s'inspirer de ce découpage. L'idée proposée pour obtenir des méthodes souples d'emploi et évolutives aisément dans le temps est le suivant :

- initialiser un blob dans lequel nous allons écrire le contexte du graphe,
- construire le blob suivant un principe permettant l'évolutivité,
- proposer des méthodes permettant simplement de modifier et lire ce blob,
- utiliser le blob pour tracer le graphe.

Initialiser le blob consiste à préparer un blob avec à l'intérieur toutes les valeurs par défaut permettant de tracer un graphe même si l'utilisateur ne change aucun paramètre. Pour cela nous écrivons la méthode GK_Init.

```
C_BLOB(gk_blob_parameter)

FIXER TAILLE BLOB(gk_blob_parameter;0)
GK_Manage_Parameter ("Read")
GK_Manage_Parameter ("Write")
```

Dans cette méthode nous commençons par déclarer le blob, et fixer sa taille à zéro octect.
Ensuite nous passons deux appels à la méthode GK_Manage_Blob.
Cette méthode est une méthode à points d'entrée multiples permettant d'utiliser le blob de manière transparente sans être obligé de connaître sa structure. L'appel GK_Manage_Blob(« Read ») permet de lire les valeurs contenu dans le blob. Comme ici il n'y a encore aucune valeur, la méthode définie les valeurs par défaut. L'appel GK_Manage_Blob(« Write ») écrit dans le blob les valeurs qui viennent d'être définies.
La lecture de la méthode GK_Manage_Blob montre comment écrire une structure évolutive au sein d'un blob. Effectivement si entre le moment ou le blob est écrit et le moment ou celui-ci est relu, la structure peut avoir évolué par des modification effectuée dans le code. Dans ce cas la structure Si (ok =1) … Fin de si, Si (ok=0) … Fin de si , permet de contrôler la validité des informations dans le temps.

```
C_ALPHA(20;$1)

Au cas ou
 : ($1="Read")
   $offset:=0
   BLOB VERS VARIABLE(gk_blob_parameter;gk_x_center;$offset)
   Si (ok=0)
     gk_x_center:=200
   Fin de si
   BLOB VERS VARIABLE(gk_blob_parameter;gk_y_center;$offset)
   Si (ok=0)
     gk_y_center:=200
   Fin de si
   BLOB VERS VARIABLE(gk_blob_parameter;gk_main_radius;$offset)
   Si (ok=0)
     gk_main_radius:=150
   Fin de si
   BLOB VERS VARIABLE(gk_blob_parameter;gk_start_angle;$offset)
   Si (ok=0)
```

```
      gk_start_angle:=90
    Fin de si
    BLOB VERS VARIABLE(gk_blob_parameter;gk_rotation;$offset)
    Si (ok=0)
      gk_rotation:=-1
    Fin de si
    BLOB VERS VARIABLE(gk_blob_parameter;gk_scale_steep;$offset)
    Si (ok=0)
      gk_scale_steep:=1
    Fin de si
    BLOB VERS VARIABLE(gk_blob_parameter;$nb_col;$offset)
    BLOB VERS VARIABLE(gk_blob_parameter;$nb_ligne;$offset)
    Si (ok=0)
      $nb_col:=1
      $nb_ligne:=5
      TABLEAU REEL(gk_values;$nb_col;$nb_ligne)
      Boucle ($j;1;$nb_ligne;1)
        gk_values{1}{$j}:=$j
      Fin de boucle
    Sinon
      TABLEAU REEL(gk_values;$nb_col;$nb_ligne)
      Boucle ($i;1;$nb_col;1)
        BLOB VERS VARIABLE(gk_blob_parameter;$_tempo;$offset)
        Si (ok=1)
          COPIER TABLEAU($_tempo;gk_values{$i})
        Sinon
          Boucle ($j;1;$nb_ligne;1)
            gk_values{1}{$j}:=$j
          Fin de boucle
        Fin de si
      Fin de boucle
    Fin de si

  : ($1="Write")
    $offset:=0
    VARIABLE VERS BLOB(gk_x_center;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB(gk_y_center;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB(gk_main_radius;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB(gk_start_angle;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB(gk_rotation;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB(gk_scale_steep;gk_blob_parameter;$offset)
     $nb_col:=Taille tableau(gk_values)
     $nb_ligne:=Taille tableau(gk_values{1})
    VARIABLE VERS BLOB($nb_col;gk_blob_parameter;$offset)
    VARIABLE VERS BLOB($nb_ligne;gk_blob_parameter;$offset)
    Boucle ($i;1;$nb_col;1)
      COPIER TABLEAU(gk_values{$i};$_tempo)
      VARIABLE VERS BLOB($_tempo;gk_blob_parameter;$offset)
    Fin de boucle

Fin de cas
```

Maintenant que nous savons initialiser notre blob, nous pouvons commencer à l'utiliser. Pour cela nous traçons dans un formulaire une zone 4d Chart nommée par exemple gk_zone_demo.
Dans un bouton au label « dessin », nous initialisons le blob en passant la ligne GK_Init.

Il nous faut à présent définir les valeurs à tracer. Pour cela nous écrivons (à titre de démonstration) des tableaux de valeurs directement au sein du bouton. Vous pourrez bien sur récupérer les valeurs depuis d'autres sources de données, mais là n'est pas notre sujet !
Pour passer les valeurs au graphe nous utilisons à nouveau le blob de stockage des paramètres car somme toute les valeurs ne sont qu'un des paramètres du dessin.
Dans l'exemple fourni avec cette note nous avons choisi de dessiner trois courbes ; il nous faux donc passer les trois tableaux de valeurs. Pour cela nous avons écrit la méthode GK_Set_Values qui admet des pointeurs de tableau en paramètres. Cette méthode rassemble tous les tableaux dans un seul tableau à deux dimensions puis demande à la méthode GK_Manage_Blob de stocker ce tableau dans le blob.

```
C_POINTEUR(${1})


$max_ligne:=0
Boucle (${i};1;Nombre de parametres)
 Si (Taille tableau(${${i}->)>$max_ligne)
    $max_ligne:=Taille tableau(${${i}->)
 Fin de si
Fin de boucle

TABLEAU REEL(gk_values;Nombre de parametres;$max_ligne)

Boucle (${i};1;Nombre de parametres)
 Boucle (${j};1;$max_ligne;1)
  Si (${j}<=Taille tableau(${${i}->))
    gk_values{${i}{${j}:=${${i}->{${j}}
  Sinon
    gk_values{${i}{${j}:=0
  Fin de si
 Fin de boucle
Fin de boucle


GK_Manage_Parameter ("Write")
```

Nous pouvons à présent nous occuper des autres paramètres du graphe. Dans la plupart des cas il suffit de changer une valeur qui sera stockée dans le blob. Si la tentation est grande de le faire directement, nous vous le déconseillons car à terme le code sera confus et moins maintenable. Nous préférons utiliser une méthode qui se charge de modifier la valeur. Cette méthode amène de la lisibilité au code mais permet aussi une modification de gestion de la valeur sans que les codes d'appel soient à modifier également. Cette méthode est nommée GK_Set_Parameter et admet deux paramètres.

```
C_ENTIER($1)    `le parametre ...
C_REEL($2)    `la valeur

$param:=$1
$valeur:=$2
Au cas ou
 : ($param=kGK_x_center )
    gk_x_center:=$valeur
```

```
  : ($param=kGK_y_center )
    gk_y_center:=$valeur

  : ($param=kGK_main_radius )
    gk_main_radius:=$valeur

  : ($param=kGK_start_angle )
    gk_start_angle:=$valeur

  : ($param=kGK_rotation )
    gk_rotation:=$valeur

  : ($param=kGK_scale_steep )
    gk_scale_steep:=$valeur


Fin de cas

GK_Manage_Parameter ("Write")
```

Le premier paramètre défini la valeur ciblée, et le second paramètre défini le contenu de cette valeur. Dans notre exemple nous avons choisi de définir les valeurs ciblées via des constantes personnalisées (pour savoir comment définir ces propres constantes personnalisées voir la note technique de XXXXXXXXXXXXXXXX). Le code de la méthode GK_Set_Parameter est relativement simple, et est près à évoluer en fonction des besoins c'est-à-dire au fur et à mesure de la définition de nouveaux paramètres modifiable pour le graphe. Nous en profitons pour écrire la méthode symétrique GK_Get_Parameter qui permet de connaître le contenu d'une valeur ciblée. Il est important d'écrire dans un même temps les méthodes Set et Get afin d'offrir à votre utilisateur le contrôle complet du blob.

```
C_ENTIER($1)    `le parametre ...
C_REEL($0)    `la valeur

$param:=$1
GK_Manage_Parameter ("Read")

Au cas ou
  : ($param=kGK_x_center )
    $0:=gk_x_center

  : ($param=kGK_y_center )
    $0:=gk_y_center

  : ($param=kGK_main_radius )
    $0:=gk_main_radius

  : ($param=kGK_start_angle )
    $0:=gk_start_angles

  : ($param=kGK_rotation )
    $0:=gk_rotation
```

```
  : ($param=kGK_scale_steep )
    $0:=gk_scale_steep



Fin de cas
```

Nous sommes à présent prêt à tracer notre graphe. Pour cela nous avons défini la méthode GK_Draw_Graph qui reçoit en paramètre la zone qui accueillera le graphe. La lecture de la méthode doit vous être familière après la lecture du début de cette note. Vous noterez que les valeurs contenues dans le blob sont utilisées pour tracer les divers éléments du graphe.

```
C_ENTIER LONG($1)    `la zone 4D Chart


$zone_Chart:=$1

ch_EXECUTER MENU ($zone_Chart;1001)




gk_nb_graph:=Taille tableau(gk_values)
gk_nb_values:=Taille tableau(gk_values{1})
$Max:=0
Boucle ($graph;1;gk_nb_graph;1)
  Boucle ($Point;1;gk_nb_values)
    Si ($Max<gk_values{$graph}{$Point})
      $Max:=gk_values{$graph}{$Point}
    Fin de si
  Fin de boucle
Fin de boucle
$Max:=Ent($Max)+1

TABLEAU ALPHA(20;_legende_serie;gk_nb_values)
_legende_serie{1}:="Résolution"
_legende_serie{2}:="Rendu couleur"
_legende_serie{3}:="Sensibilité"
_legende_serie{4}:="Flash"
_legende_serie{5}:="Autofocus"
_legende_serie{6}:="Sensibilité"

TABLEAU REEL($_angle;gk_nb_values)

$angle_depart:=gk_start_angle*Degré
Boucle ($Point;1;gk_nb_values)
  $_angle{$Point}:=$angle_depart+(360*Degré /gk_nb_values*($Point-1)*gk_rotation)
Fin de boucle


TABLEAU REEL($_x;gk_nb_values+1)
TABLEAU REEL($_y;gk_nb_values+1)
```

```
Boucle ($ligne;1;$Max;gk_scale_steep)
   $rayon_ligne:=gk_main_radius*$ligne/$Max
  Boucle ($Point;1;gk_nb_values)
    $_x{$Point}:=gk_x_center+(Cos($_angle{$Point})*$rayon_ligne)
    $_y{$Point}:=gk_y_center-(Sin($_angle{$Point})*$rayon_ligne)
  Fin de boucle
   $_x{gk_nb_values+1}:=$_x{1}
   $_y{gk_nb_values+1}:=$_y{1}
    $poly:=ch_Tableau vers polygone ($zone_Chart;$_x;$_y)
  ch_FIXER TRAMAGE ($zone_Chart;$poly;1;-1)
Fin de boucle

Boucle ($graph;1;gk_nb_graph;1)
  Boucle ($Point;1;gk_nb_values)

$_x{$Point}:=gk_x_center+(Cos($_angle{$Point})*gk_main_radius*gk_values{$graph}{$Point}/$Max)
                                                                    $_y{$Point}:=gk_y_center-
(Sin($_angle{$Point})*gk_main_radius*gk_values{$graph}{$Point}/$Max)
  Fin de boucle
   $_x{gk_nb_values+1}:=$_x{1}
   $_y{gk_nb_values+1}:=$_y{1}
    $polygone:=ch_Tableau vers polygone ($zone_Chart;$_x;$_y)
  ch_FIXER TRAMAGE ($zone_Chart;$polygone;3;ch_Index vers couleur ($graph+1))
Fin de boucle

Boucle ($Point;1;gk_nb_values)
                                                    $ligne:=ch_Creer trait
($zone_Chart;gk_x_center;gk_y_center;gk_x_center+(Cos($_angle{$Point})*gk_main_radius);gk_x_cen
ter-(Sin($_angle{$Point})*gk_main_radius))
Fin de boucle

Boucle ($ligne;1;$Max;gk_scale_steep)
  $Point:=1
    $rayon_ligne:=gk_main_radius*$ligne/$Max
   $x:=gk_x_center+(Cos($_angle{$Point})*$rayon_ligne)
   $y:=gk_y_center-(Sin($_angle{$Point})*$rayon_ligne)
    $texte:=ch_Creer texte ($zone_Chart;$x-20;$y+5;$x-2;$y+15;Chaine($ligne))
  ch_FIXER TEXTE ($zone_Chart;$texte;ch_Numero de police ("Geneva");9;Gras ;ch_Index vers
couleur (16);2)
Fin de boucle

Boucle ($Point;1;gk_nb_values)
   $x:=gk_x_center+(Cos($_angle{$Point})*gk_main_radius)
   $y:=gk_x_center-(Sin($_angle{$Point})*gk_main_radius)
  Si ($x>=gk_x_center)
     $x1:=$x+10
     $x2:=$x+100
     $justif:=0
  Sinon
     $x1:=$x-10
     $x2:=$x-100
     $justif:=2
  Fin de si
  Si ($y>=gk_y_center)
    $y1:=$y+5
    $y2:=$y+15
```

```
  Sinon
    $y1:=$y-5
    $y2:=$y-15
  Fin de si
   $texte:=ch_Creer texte ($zone_Chart;$x1;$y1;$x2;$y2;_legende_serie{$Point})
    ch_FIXER TEXTE ($zone_Chart;$texte;ch_Numero de police ("Geneva");9;Normal ;ch_Index
vers couleur (16);$justif)
Fin de boucle


ch_FIXER PROPRIETES ($zone_Chart;-1;0;-1;0)
```

Comme l'apprentissage par l'exemple est une très bonne formation, nous avons laissé volontairement plusieurs éléments non paramétrables au sein du graphe. A vous de modifier les méthodes (ou d'en ajouter) pour obtenir un module complet. Par exemple il vous faudra ajouter un paramètre pour gérer le remplissage ou non des polygones (agir sur la commande ch FIXER TRAMAGE). Il vous faudra ajouter une commande GK_Set_Legende afin de définir les légendes de votre graphe.

A travers cet exemple nous avons essayer de vous montrer qu'il est possible de construire des graphes très personnalisés grâce à 4D Chart, et nous espérons vous avoir donner des pistes pour rendre générique et agréable d'utilisation votre module. A vous à présent de faire jouer votre imagination pour en tirer le meilleur parti et proposer ainsi à vos utilisateurs des graphes percutants !