graphLib

un ensemble de routines pour produire un graphique en Applescript

Guide d'utilisation

G. Coray

mars 2022

Introduction

GraphLib est un ensemble de routines de tracé graphique utiles dans une application Applescript.

Dans toute application, la présentation de données ou de resultats à l'utilisateur requiert soit une interface graphique ou la conversion de données en texte clair. Comme "une image vaut mille paroles" graphLib propose des fenêtres et des fichiers graphiques comme moyen de communiquer des resultats à l'utilisateur. Il utilise pour cela un logiciel grahique pilotable en Applescript[1].

Les applications typiques réalisées avec graphLib sont un grapheur pour représenter des fonctions ou courbes paramétrées, un illustrateur de statistiques (histogramme, camembert) un manipulateur de Rubicube en 3D et un interprète de Systèmes fractals de Lindenmayer.[2]

Une motivation importante pour collectionner des primitives de base sous la forme d'une bibliotèque de routines est l'économie d'efforts pour rechercher les détails des commandes graphiques dans les librairies ou applications graphiques disponibles, pilotables en Applescript.

Le style adopté ici est proche du paradigme procédural que l'on retrouve dans Algol, Pascal, Python ou C. Chaque routine a un nombre fixe de paramètres positionnels, faciles à documenter. Comme dans la plupart des librairies graphiques le nombre de paramètres nécessires dans chaque routine peut être encombrant, dans un petit nombre de cas graphLib fait usage de variables globales comme paramètres optionnels.

Ce Guide graphLib est organisé en quatre parties:

1. Chargement et initialisations
2. Décorations
3. Tracé des éléments graphiques
4. Affichage et sauvegarde

---

[1] *Adobe Photoshop 2021* pour la version courante. Les versions plus anciennes de graphLib sont basées sur *Photoshop CS, Photoshop 2018* resp. *Smile* de Satimage.Fr. Ce guide couvre aussi ces versions. Cf. Annexe 1 *Installation*

[2] Certaines applications requièrent en outre textLib, une bibliothèque pour la manipulation des textes, listes et expressions arithmétiques.
Deux variantes à usage pédagogique de la graphLib de base sont disponibles: TurtleLib and EuclidLib. TurtleLib permet de programmer en *Logo*, le langage proposé par Seymour Papert pour l'initiation à l'algorithmique, les itinéraires d'une tortue virtuelle. EuclidLib présente un plan Euclidien pour construire des figures géométriques "à la règle et au compas" et observer leurs propriétés telles que l'alignement sur la droite d'Euclide ou le théorème de Thalès.

1. Chargement et initialisations

Le chargement

L'application que l'on désire "scripter" (nous dirons programmer ou écrire) doit charger la bibliothèque "graphLib.scpt" pour accéder à ses routines dans le nouveau script. Elle sera désignée par un nom interne, par exemple lib ou graphlib. Cette opération s'énonce:

```
set graphlib to (load script file "Macintosh
        HD:Library:Scripts:Libs:graphLib.scpt")
```

Note: on suppose qu'une copie compilée de graphLib.script se trouve dans "Macintosh HD:Library:Scripts:Libs". Pour plus de détails cf. Annexe 1.

A partir de là toutes les routines exportées par graphlib sont accessibles dans la portée des instructions **tell** graphlib. Voici un exemple schématique de script d'application:

```
set graphlib to (load script file "Macintosh
        HD:Library:Scripts:Libs:graphLib.scpt")
tell graphlib
      CartesianDoc(-3, 3, 0, 1, "A4") -- crée le graphique
      disc({0,0},1.5,"R")    -- appels de routines graphiques (ici un rond)
      saveinfile("Rond.jpg") -- sauvegarde des résultats dans un fichier
end tell
```

Note**:** Sur chaque ligne, le texte qui suit "--" est un commentaire.

L'initialisation par CartesianDoc

CartesianDoc(a, b, ymedian, anamorphose, Titel)

C'est la première routine à appeler dans tout script qui produit un graphique ou une animation. Elle crée un graphique (objet de l'application graphique sous-jacente) et ouvre une fenêtre pour afficher son contenu. Toutes les autres routines de graphLib se réfèrent implicitement à cette fenêtre. Le graphique est automatiquement muni d'un système de coordonnées cartésien Oxy utilisé pour positionner les figures à tracer par les routines détaillées ci-dessous au §3.

Les paramètres de CartesianDoc:

Les paramètres a, b, ymedian, anamorphose, Titel ont la signification suivante:

Titel est le nom porté par la fenêtre.

a, b: l'abscisse x prend ses valeurs dans l'intervalle [a, b]

ymedian est l'ordonnée du centre de la fenêtre graphique et

anamorphose donne le rapport d'échelle verticale/horizontale.
Ce choix s'avère plus pratique que, par exemple, la donnée de l'intervalle des ordonnées affichables pour y.

Par exemple, CartesianDoc(-15, 15, 0, 1, "A4") affiche une fenêtre intitulée "A4" dont les points ont des coordonnées comprises entre -15 et +15 pour x et de -10.5 à +10.5 environ pour y.
Imprimée sur un feuille A4 (orientée en mode "paysage") la figure affichée dans la fenêtre occupera la feuille imprimée.

Un autre exemple. Supposons que le but est un graphique de statistiques sur 20 ans où toutes les valeurs à afficher sont des pourcentages. On aura sur l'axe horizonal des valeurs extrêmes de -0.5 à 20.5, avec une marge de 0.5 pour éviter le débordement. Verticalement les valeurs sont comprises entre 0 et 100 (soit 5 fois plus denses que les abscisses) à reporter sur une feuille moins haute que large (env. 0.7). On fera par conséquent CartesianDoc(-0.5, 20.5, 100/2, 1/5 * 0.7, "Stats [%]").

Les paramètres optionnels

Le paramètre optionnel margins se présente comme une variable globale initialisée avec la valeur par défaut à 2 [pourcent]. C'est l'espace vierge bordant la fenêtre où il n'y aura pas de tracé.
Pour modifier ce choix il suffit d'affecter la variable margins, avant d'appeler CartesianDoc, avec le pourcentage souhaité.
Par exemple, **set** margins **to** 0 supprimera l'espace vierge qui borde la fenêtre sur les 4 côtés: la feuille sera imprimée jusqu'au bord.
Deux autres paramètres optionnels, docWidth et docHeight, permettent d'ajuster la taille de la fenêtre affichée et la résolution (peu utile si le résultat est conservé en format .pdf) voire de changer la proportion largeur/hauteur. Par exemple {docWidth, docHeight}={1000, 1414} produira un graphique au format A4 "portrait".
Enfin, marksOx, marksOy et rulesOx, rulesOy forcent le dessin des axes de coodonnées et des filets par Axes et Grid (voir §2 ci-dessous)

Les constantes exportées

L'appel à CartesianDoc a également pour effet d'initialiser les constantes:
global xmin, xmax, ymin, ymax, pica, dtp
Elles fournissent au programmeur l'information pour disposer ses figures dans la page. xmin, xmax sont identiques aux paramètres donnés a,b, alors que ymin, ymax sont calculés en fonction des autres paramètres.

Les constantes typographiques pica, dtp permettent au programmeur de disposer et mesurer du texte, par exemple pour justifier une légende près d'une figure, ou par rapport au repère des coordonnées.
pica est la hauteur standard d'un caractère en corps 12 (imprimé su A4)
dtp vaut un *point* (standard Digital Typography Point) soit 1/12 de pica.

Une longueur exprimée en dtp ou en multiples de pica aura toujours la même représentation "sur papier A4", indépendamment des échelles introduites par le système cartésien, soit env. 4 [mm] pour 1 pica.

Note. Un appel à CartesianDoc doit figurer dans tout script destiné à produire des documents graphiques ou des animations.

2. Les décorations Axes, Grid et Fill

L'initialisation par CartesianDoc nous place devant une fenêtre vide. Elle est souvent complétée par l'ajout de repères ou d'un fond de couleur. Nous allons dès maintenant aborder ces utilitaires de "décoration": Axes, Grid et Fill.

Axes

Axes(marksOx, marksOy) dessine les axes du repère cartésien avec des marques ou échelons numériques équidistants. Le nombre de repères tracés sur l'axe de coordonnées Ox est donné par le paramètre marksOx. Cette valeur est indicative, elle sera arrondie afin de placer les repères aux multiples d'une puissance de 10 adéquate. Une valeur nulle supprime le tracé des axes.

Grid

Grid(rulesOx, rulesOy) forme une grille qui quadrille le plan. Les paramètres rulesOx, rulesOy fixent le nombre minimum de filets verticaux, resp. horizontaux de la grille dessinée. Une valeur nulle supprime le tracé de la grille.

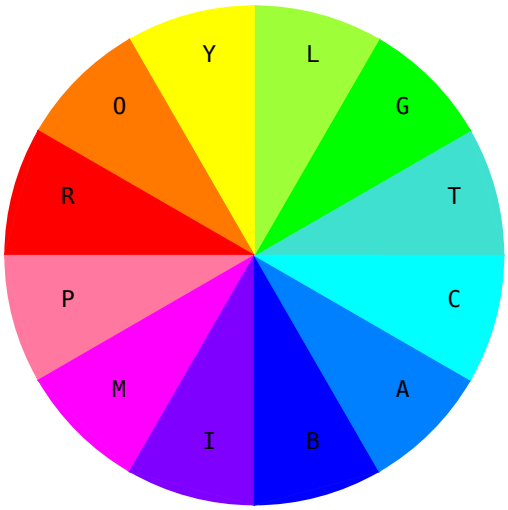
Fill

Fill(colour) permet de donner une couleur de fond à la fenêtre (et aux documents produits). Le paramètre colour suit les conventions de couleur RGBA ci-dessous. A défaut d'un appel à Fill le fond restera transparent, ce qui apparaîtra utilement dans les fichiers ou diapos produits en format .png (mais pas en .jpg ni en .pdf où le fond est blanc).

Les couleurs

La notion de couleur mérite un chapitre en soi, saisissons l'occasion pour en rappeler l'essentiel. Afin de faciliter la programmation colour peut être spécifiée de diverses façons.
La méthode facile est de choisir dans la *palette* suivante:

Palette



C	Cyan
A	Azur
B	Blue-bleu
I	Indigo-violet
M	Magenta
P	Pink-rose
R	Red
O	Orange
Y	Yellow-jaune
L	Lime
G	Green-vert
T	Turquoise

qu'il faut encore à compléter par les caractères "W", "N", "-" ,"*" et espace:

"W" , "White" ou "blanc" "N", "K" , "blacK" ou "Noir"
"-" , "gris" ou "gray" "*" ou "random" couleur aléatoire
" " ou "transparent" pour un tracé invisible

Plus généralement la couleur est une valeur qui peut être donnée
- soit par un code contenant six *caractères hexadécimaux*, comme p. ex. "#FF0000" pour le rouge,
- soit par le triplet *RGB* comme p.ex. {255,0,0} pour les trois valeurs R=255, G=0, B=0 où on reconnaît les composantes *Red*, *Green* et *Blue*
- soit enfin avec la notation *RGBA32* à quatre composantes qui ajoute *A* ou *alpha* pour *l'opacité*, le complémentaire de la transparence, aux trois couleurs fondamentales R, G et B. Dans ce cas les valeurs des quatre composantes sont normalisées à l'intervalle d'entiers [0, 255].
Par exemple Fill("W") , Fill({255,255,255}) ou Fill({255,255,255,255}) produiront un fond blanc, tandis que Fill("K") ou Fill(#000000") donne un fond noir. Notons que que Fill(" ") ou Fill({0,0,0,0}) n'aura aucun effet puisque *alpha*=0 signifie transparence. Avec une composante *alpha* comprise entre 0 et 255 Fill peut être utile pour estomper, voire effacer, une figure déjà produite.

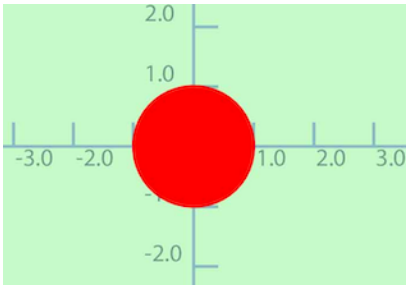
Note. La conversion de ces différents codes de couleur vers la norme RGBA32 est effectuée par la routine rgb, appliquée automatiquement à tous les paramètres de couleur apparaissant dans les routines de tracé de graphLib telles que Fill. Elles ne sera donc pas détaillée davantage.

3. Le tracé des éléments graphiques.

Un graphique construit à l'aide de graphLib peut contenir des éléments de divers types: disque, segment, flèche, rectangle, polygone, courbe ou arc de cercle. Chaque élément est superposé aux précédents lors de l'appel à une routine de tracé. Voici le détail des routines ad hoc.

disc, dot

disc(centre, rayon, couleur) dessine un disque dont le centre est donné par une paire {xc,yc} de nombres réels, ses coordonnées xc, yc.
Par exemple, on place un rond rouge à l'origine à l'aide de
disc({0,0}, 1 , "R") -- rayon de 1 unité (soit 1 cm si xmax-xmin~30).



Note: disc suppose la même échelle sur les axes Oy et Ox. Si au contraire anamorphose ≠ 1 on obtient un rond elliptique aplati.

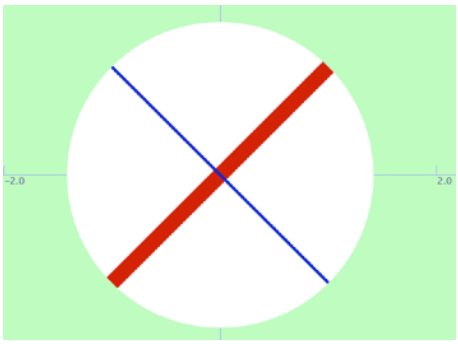
dot(centre, diam, couleur) pallie ce défaut en dessinant un rond circulaire de diamètre diam exprimé en points dtp (à l'instar du corps d'un texte).
L'instruction suivante produit le même effet que l'exemple précédent:
dot({0,0}, 60 , "R") -- diamètre du point = 60 dtp = 2 cm sur A4

segment, arrow

segment(P0, P1, epaisseur, couleur) trace un segment de droite entre les points P0 et P1. On spécifie l'epaisseur du trait en multiples de dtp. En réalité ce que l'on obtient est un rectangle oblique de longueur égale à la distance de P0 à P1 et de largeur égale à epaisseur (en dtp).

Exemple: deux diamètres d'un disque, avec des épaisseurs différentes

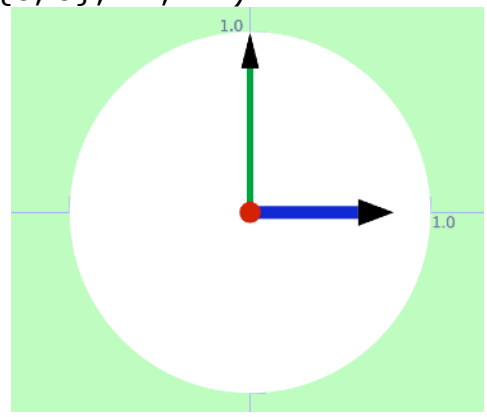
```
disc({0, 0}, sqrt(2), "W")
segment({-1, -1}, {1, 1}, 20 , "M")
segment({-1, 1}, {1, -1}, 4 , "B")
```



arrow(P0, P1, epaisseur, couleur, headLength, headWidth, headColor) trace un segment dont l'extrémité est décorée d'une pointe de flèche. Dimensions et couleur de la pointe sont détaillés dans headLength, headWidth en dtp, et headColor. Bien entendu la pointe doit être plus courte que le segment entier, qu'elle recouvre en partie; elle sera automatiquement raccourcie au besoin.

Exemple

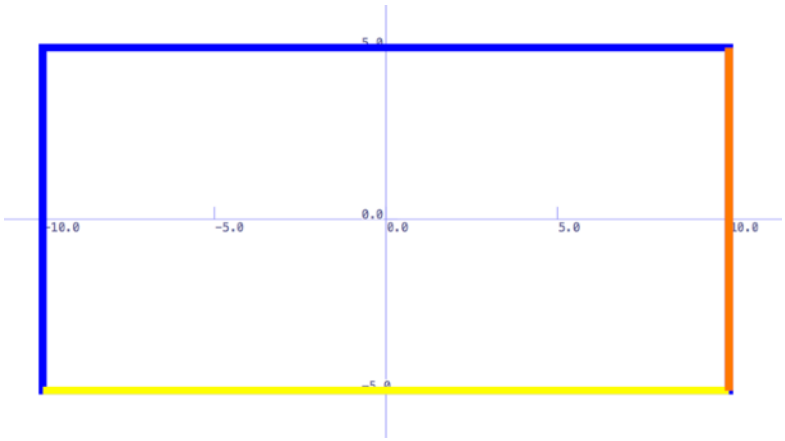
```
disc({0, 0}, 1, "W")                              -- cadran blanc
arrow({0, 0}, {0, 1}, 5 , "G", 20, 10, "K")       -- minutes
arrow({0, 0}, {0.8, 0}, 10 , "B", 20, 15, "K")    -- heures
dot({0, 0}, 12 , "R")                             -- centre
```



rectangle

rectangle(P0, P1, epaisseur, couleur) dessine un rectangle dont les côtés sont parallèles aux axes. P0 et P1 sont deux sommets opposés, en diagonale.
L'exemple suivant, et le tracé qu'il produit,

```
rectangle({-10, -5}, {10, 5}, 8 , "B")
segment({-10, -5}, {10, -5}, 8 , "Y")
segment({ 10, -5}, {10,  5},  8 , "O")
```



permettent d'observer le détail de la couverture des angles, plus propre dans le rectangle (bleu) qu'avec les segments (jaune et orange) superposés[3].

Le paramètre epaisseur

Dans l'exemple précédent l'epaisseur est 8 [dtp], soit env. 2.7 mm pour un graphique imprimé au format A4. En effet l'unité typographique du point dtp, équivaut physiquement (c.à.d.sur papier A4) à env. 1/3 mm, indépendamment des échelles et unités utilisées pour les coordonnées. Une convention particulière est adoptée pour la valeur limite epaisseur=0 du paramètre. Elle est utilisée pour obtenir un *rectangle plein* de la même couleur. Ceci est illustré dans l'exemple

```
rectangle({-2, -1.7}, {2, 1.7}, 0, "R")       -- toile rouge
rectangle({-0.5, -1.4}, {0.5, 1.4}, 0, "W") -- croix blanche 1/2
rectangle({-1.4, -0.5}, {1.4, 0.5}, 0, "W") -- croix blanche 2/2
```

Notons encore qu'un rectangle plein peut aussi être obtenu à l'aide de segment; ainsi rectangle({0, 0}, {a, b}, 0, couleur) est équivalent à segment({0,b/2}, {a,b/2}, b/dtp , couleur).

polyline, polygon

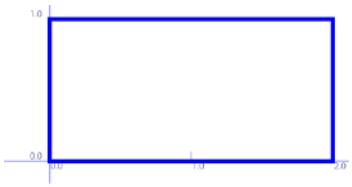
Polyline(listeDesSommets, rayon, epaisseur, couleur) trace une ligne brisée formée de segments reliant deux points consécutifs de la

[3] Le rectangle peut être dessiné correctement avec quatre segments si on prolonge ces derniers d'une demi-épaisseur à chaque extrémité.

listeDesSommets. Une attention particulière est portée au raccord des segments, contrôlé par le paramètre rayon, à chaque sommet sauf aux extrémités.

Ainsi, pour un rayon nul, les exemples suivants donnent le même résultat

```
rectangle({0, 0}, {2, 1}, 5 , "B") -- respectivement:
set sommets to {{0, 1},{0, 0}, {2, 0}, {2, 1}, {0, 1},{0, 0}}[4]
Polyline(sommets, 0, 5 , "B")
```



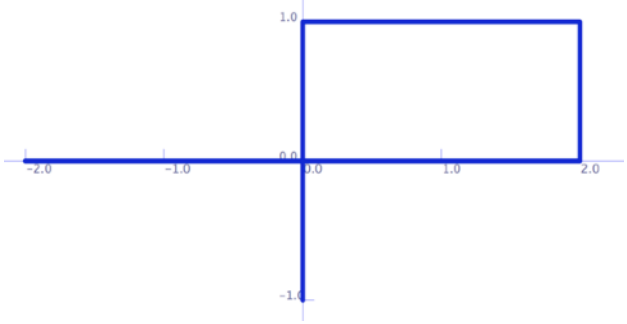
Le paramètre epaisseur

Le paramètre epaisseur a le même rôle que pour le rectangle, il est donné en unités dtp.

Le paramètre rayon

Pour illustrer l'emploi du paramètre rayon avec des valeurs non nulles, considérons le parcours:

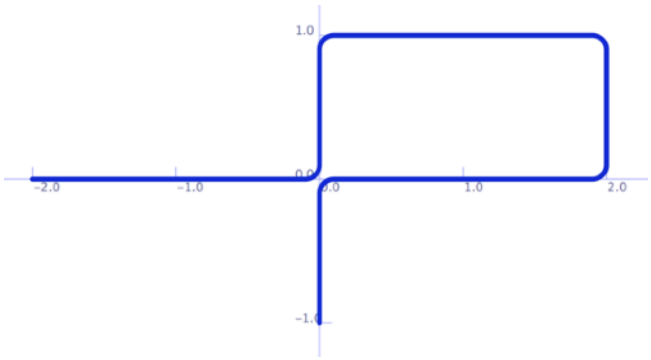
```
set parcours to {{-2,0}, {0,0}, {0,1}, {2,1}, {2,0}, {0,0}, {0,-1}}
polyline(parcours, 0, 5, "B")
```



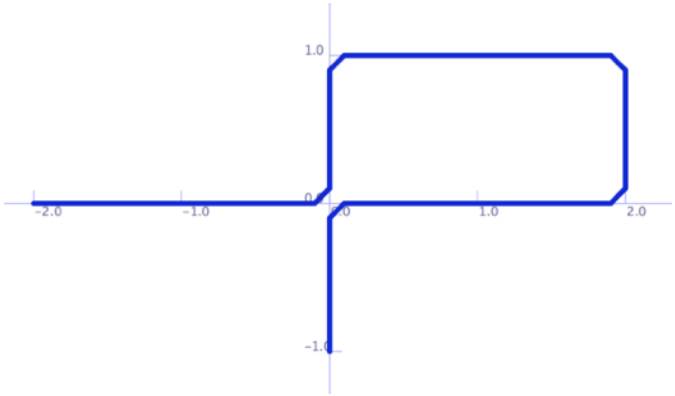
La figure obtenue ne permet pas de deviner le sens du parcours. Pour remédier à ce défaut on peut arrondir les angles du polygone à chaque sommet.

C'est le rôle du paramètre rayon: avec une valeur de 15 on obtient ainsi

[4] On note le segment redondant entre {0,0} et {0,1} destiné à compléter les coins.



Une valeur négative du rayon fait apparaître un chanfrein au lieu de l'arrondi



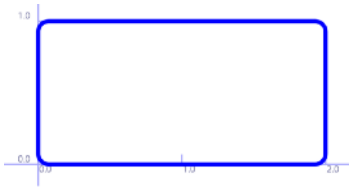
Cas des polygones fermés.

Un polygone est une ligne brisée *fermée*, donnée par ses sommets. Les sommets consécutifs sont reliés par des segments et le dernier sommet est relié au premier.

Polygon(listeDesSommets, rayon, epaisseur, couleur) trace une ligne fermée dont tous les sommets sont traîtés de la même manière, un segment étant ajouté entre le dernier et le premier sommet de la liste. Dans ce cas, avec rayon≠0, l'arrondi (resp. le chanfrein) s'applique également à ces deux sommets.

L' exemple suivant produit le rectangle aux quatre coins arrondis:

```
set sommets to {{0, 0}, {0, 1}, {2, 1}, {2, 0}}
Polygon(sommets, 15, 5 , "B")
```

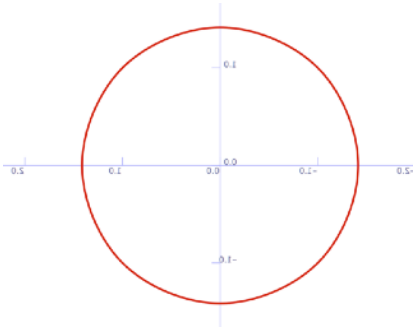


Note. La longueur des segments doit être au moins le double du rayon, (sinon le rayon effectif sera automatiquement adapté).
Comme pour le rectangle, lorsque epaisseur=0 le polygone sera rempli.

circle

Bien que polyline et polygon permettent en principe l'approximation à n'importe quelle courbe il est pratique de disposer des deux primitives spécialisées arc et circle.
circle(centre, rayon, epaisseur, couleur) trace un cercle de rayon donné, centré en centre. L'epaisseur est donnée en dtp; par analogie avec la convention ci-dessus, lorsque epaisseur=0, l'effet est le même que disc(centre, rayon, couleur).
Exemple:

```
set r to 2^0.5  -- sqrt (2)
circle({0, 0}, r , 3 , "R")
```



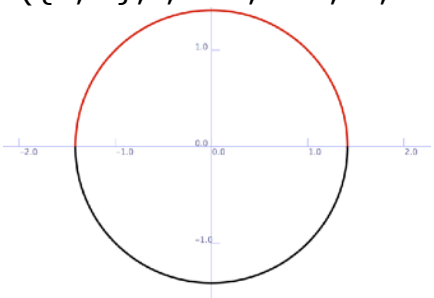
Notons que ce cercle peut aussi être obtenu comme un carré aux coins fortement arrondis:

```
set carreCiconscrit to {{-r,-r},{r,-r},{r,r},{-r,r}}
set rayon to  r/dtp -- ou toute valeur plus grande
polygon(carreCiconscrit, rayon, 3,"R")
```

arc

arc(centre, rayon, alpha, beta, epaisseur, couleur) trace un arc de cercle délimité par les angles alpha et beta. Si on choisit alpha=0 et beta=360 (ou simplement alpha=beta) on obtient un cercle complet. L'exemple suivant illustre le cas de deux demi-cercles qui se complètent.

```
set r to 2^0.5  -- sqrt (2)
arc ({0, 0}, r, 0, 180, 3 , "R")
arc ({0, 0}, r, 180, 360, 3 , "K")
```



Enfin, par analogie avec la convention ci-dessus, lorsque epaisseur=0, arc dessine un secteur de cercle remplissant l'espace entre le centre et l'arc de cercle; pratique pour les "camemberts" (pie-charts).

Ainsi, arc(centre, r, 0, 360, 0, couleur) est équivalent à disc(centre, r, couleur).[5]

Paramètres optionnels rares

Le rendu de Arc resp. Cercle est obtenu par un polygone régulier approximant. La densité des points est déterminé par la variable globale tolerance, fixée par défaut à dtp/6, une fraction de pixel. Deux points à une distance inférieure à tolerance ne sont pas traîtés comme distincts! Un autre global, contourMode, permet de flouter les bords (≠0) voire de remplir l'extérieur (<0) des disques ou polygones. Il s'exprime en dtp, son effet dépend de l'application.

txt - labels et légendes

txt(legend, position, textsize, textColor) affiche le texte de legend à la position indiquée.
position: coin inférieur gauche de la zone occupée par le texte affiché.
textsize: corps des caractères, souvent 12 (unité=dtp) soit un pica.
L'exemple suivant place "Page 2" dans le coin en bas à gauche:

```
txt("Page 2", {xmin, ymin}, 12, "K")
```

txtWidth

La justification du texte d'une légende ou d'un titre nécessite parfois le calcul de la place occupée par ce dernier.
txtwidth(legend, textsize) fournit la largeur du texte de legend en unités de l'axe Ox. Par exemple pour placer "Page 3" en corps 12 dans le coin en bas à droite on ajustera:

```
txt("Page 3", {xmax-txtwidth("Page 3", 12), ymin}, 12, "K")
```

Paramètres optionnels

Signalons, pour être complets, deux variables globales, textFont, et textBackGround, qui peuvent "paramétrer" le rendu de legend.
textFont est initialisée à "Menlo" et peut être modifiée au besoin.
textBackGround est "transparent", le texte se superpose au graphique. Avec une couleur de fond plus contrastée à l'endroit du texte on peut gagner en lisibilité.

Utilitaires[6]

D'autres moyens de "mesurer" les figures peuvent s'avérer utiles lorsque anamorphose=1, à savoir distance et angle.
distance(P0,P1) fournit la distance entre deux points P0,P1, en unités du système cartésien.

---

[5] Disc, circle et arc supposent une même échelle en y et en x (anamorphose=1)

[6] graphlib met à disposition les fonctions élémentaires ln, exp, sqrt, abs, signe, cos, sin, atan. Les unités d'angle sont par défaut le degré, modifiable par cos_sin_units("rad")

angle(S0,S1) est l'angle entre deux segments, orienté de S0 vers S1. Le résultat, en degrés, est compris entre -180 et +180.

Enfin on peut obtenir un point P intermédiaire, voire extrapolé, à l'aide de: **set** P **to** interpolation(P0,P1,w) : P = (1-w)*P0 + w*P1 vectoriellement.

L'interpolation peut s'appliquer de manière générale aux nombres, points, polygones, couleurs RGBA ou autres vecteurs. Le cas particulier P0=0 est interprété comme une homothétie de rapport w et de centre P0={0,0} pour des points. C'est le produit par le scalaire w si P1 est un vecteur.

4. L'affichage et la sauvegarde

Les routines de service show, saveinfile et Diapo contrôlent l'affichage et la sauvegarde des objets graphiques créés.

show

Le rafraîchissement de la fenêtre graphique est contrôlé de manière explicite dans certains logiciels graphiques comme *Smile*:
show(0) rend la mise à jour automatique à chaque figure tracée, jusqu'à concurrence du prochain appel à show(1) ou show(-1). C'est le mode par défaut (il permet d'ignorer show).
show(1) active la fenêtre graphique, la place au premier plan du bureau et met à jour le contenu.
On attendra le prochain appel à show pour rafraichir à nouveau la fenêtre.
Lorsque la fenêtre graphique est devenue inutile et encombrante on peut s'en défaire par un show(-1).

saveInFile

saveInFile effectue la sauvegarde du graphique (l'état actuel de la fenêtre) dans un fichier de type .pdf, .jpg, .tiff ou .png.
L'appel saveInFile(filePath) conserve l'état du graphique, créé à l'aide des routines précédentes, dans un fichier repéré par le paramètre filePath. Ce dernier doit avoir la forme d'un chemin d'accès (voire d'un alias) connu par l'application Finder.
Le suffixe .pdf, .jpg, .tiff ou .png décidera du type de fichier qui en résulte. En l'absence de suffixe, .pdf est pris automatiquement. Cette valeur par défaut est recommandée parce que le fichier est léger et la définition excellente pour le graphique vectoriel produit.
Cependant l'option .png pour le suffixe a l'avantage de conserver les parties transparentes du graphique, ce qui est utile par ex. dans les animations sur fond fixe.
Une abréviation pratique: si filepath se résume à un identificateur (ne contenant pas de ":") la destination choisie sera le *bureau*.
Par ex. saveInFile("Smiley") crée ou met à jour le fichier "Smiley.pdf" sur le bureau.
Notons toutefois que la fenêtre active peut très bien être sauvegardée "à la main" pour être conservée sous la forme de fichier de type graphique.
La sauvegarde de plusieurs états graphiques dans un dossier se fera plus efficacement à l'aide de Diapo, en effet:

Diapo

Diapo() et Diapos(folderpath) servent à conserver plusieurs instantanés du graphique en progression, destinés à un *gif animé* ou à une présentation par diapositives.
Chaque appel de Diapo() ajoute un fichier à la collection, numéroté automatiquement dans l'ordre chronologique. Le dossier recevant la collection doit être créé au préalable à l'aide de Diapos(folderpath).

Les appels à Diapo utiliseront alors le nom du dossier créé par Diapos, complété par son numéro, pour chaque fichier stocké.

Diapos

Diapos(folderpath) doit être appelé avant les appels à Diapo, son paramètre folderpath désignant le dossier destination des images.
Une fenêtre de dialogue s'ouvrira alors pour permettre à l'utilisateur de
- confirmer/modifier le nom du dossier à créer pour les instantanés,
- choisir un dossier déjà existant pour y ajouter les nouvelles diapos,
- ou renoncer tout bonnement aux instantanés lors de cette exécution.

Paramètre optionnel

Les images stockées lors des appels à Diapo() sont des fichiers de type *.png* par défaut. Au besoin, ce choix peut être modifié en *.pdf* ou *.jpg* par une affectation à la variable globale DiapoFileExtension.

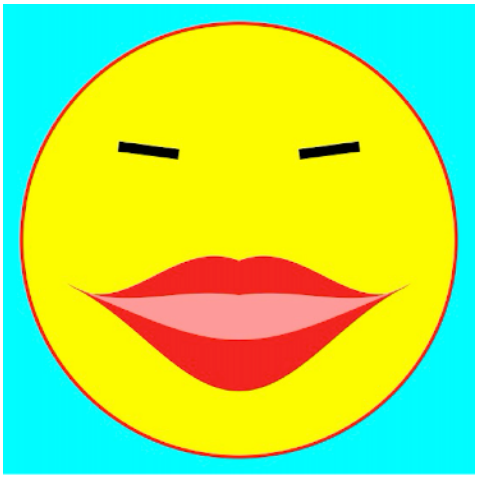
5. Un exemple complet

Voici un exemple de script compilable en une application.
But: créer une séquence d'images (aka frames) destinées au GIF animé d'un Smiley dont l'expression "souriante" varie.
Méthode: le contour de la bouche et des lèvres est dessiné au moyen de fonctions trigonométriques (coefficients testés avec l'aide de grapher).
Les paramètres sont recalculés, pour chaque image, par interpolation.

```
-- But:créer une séquence de 10 images (frames) pour l'animation d'un Smiley
global graphLib
set graphLib to load script file "Macintosh
        HD:Library:Scripts:Libs:graphLib.scpt"
set nbOFframes to 10 -- sans compter l'initiale
-----------------------------------------------------
-- paramètres pour un Smiley (valeurs numériques ou vectorielles)
set mouthcolor to {1, 0.6, 0.6, 1}  -- couleur ouverture bouche, env. "O"
set upperlip to 0.4                 -- épaisseur lèvre sup
set lowerlip to 0.3                 -- épaisseur lèvre inf
-- valeurs initiales => Param0
set lipColor0 to {0.8, 0, 0.6, 1}   -- couleur initiale des lèvres, env."M"
set uppermouth0 to 0.6              -- hauteur bouche au-dessus de l'horiz.
set lowermouth0 to -0.4             -- hauteur bouche au-dessous de l'horiz.
set mood0 to -0.6                   -- humeur = inflexion des commissures
set Param0 to {lipColor0, mouthcolor, upperlip, uppermouth0, lowermouth0,
        lowerlip, mood0} -- début
-- valeurs finales => Param1
set lipColor1 to {1, 0, 0, 1}       -- couleur finale des lèvres = "R"
set uppermouth1 to 0.1              -- id. pour diapo finale
set lowermouth1 to -0.1             -- id. pour diapo finale
set mood1 to 0.6                    -- humeur = commissures à la fin
set Param1 to {lipColor1, mouthcolor, upperlip, uppermouth1, lowermouth1,
        lowerlip, mood1} -- fin
-----------------------------------------------------
tell graphLib
    CartesianDoc(-2, 2, 0.5, 1, "Smiley")
    fill("C") -- colorer le fond
    Diapos("SmileyAnim")
    cos_sin_units("rad") -- cette fois on travaille en radians
end tell

repeat with n from 0 to nbOFframes
    Smiley(interpolation(Param0, Param1, n / nbOFframes))
end repeat
-----------------------------------------------------
```

```
on Smiley(param) -- créé une diapo (frame) selon les param. courants
      set {lipcolor, mouthcolor, upperlip, uppermouth, lowermouth, lowerlip,
                  mood} to param
      set uppermouthpts to contour(uppermouth, 1 / 2, mood)
      set lowermouthpts to contour(lowermouth, 0, mood)

      set mouthPoints to uppermouthpts & (reverse of lowermouthpts)
      set UpperLipPts to contour(upperlip + uppermouth, 1 / 2, mood) & reverse
                  of uppermouthpts
      set LowerLipPts to lowermouthpts & reverse of contour(lowermouth -
                  lowerlip, 0, mood)

      tell graphLib
            disc({0, 0.5}, 1.27, "R")                    -- bordure rouge de ...
            disc({0, 0.5}, 1.25, "Y")                    -- la face jaune

            Polygon(mouthPoints, 0, 0, mouthcolor)  -- intérieur bouche
            Polygon(UpperLipPts, 0, 0, lipcolor)         -- lèvre sup
            Polygon(LowerLipPts, 0, 0, lipcolor)         -- lèvre inf

            segment({-0.7, 1.1 - mood * 0.15}, {-0.35, 1}, 10, "K") -- oeil droit
            segment({0.35, 1}, {0.7, 1.1 - mood * 0.15}, 10, "K")    -- et gauche
            Diapo()
      end tell
end Smiley
-------------------------------------------------------
on contour(height, notch, mood) -- demi-contour (supérieur) de la bouche
      set resolution to 120 -- nb de points horizontalement
      set res to {}
      repeat with t from 0 to resolution
            set x to -1 + 2 * t / resolution   -- x de -1 à +1
            tell graphLib
                  set y to height * (sin (pi * t / resolution) ^ 2) -- ordonnée en x
                  set v to height * notch * (abs (atan (x))*4 / pi - 1)
                              -- correction au milieu pour l'échancrure en V
                  set m to 0.15 - 0.7 * mood * (-0.8 + (sqrt (0.16 + 1 - x ^ 2)))
                              -- correction aux commissures selon l'humeur
            end
            copy {x, y + v + m} to end of res
      end repeat
      return res -- vérifié expérimentalement avec grapher!
end contour
```

Résultat dans le dossier SmileyAnim: onze figures dont la neuvième ci-dessous



6. Annexes

Installation de graphLib

La bibliothèque graphLib.scpt est un script compilé à l'aide de l'éditeur de scripts, et doit se trouver dans le dossier Libs:, lui-même un élément du dossier scripts: Il est accessible via le chemin "Macintosh HD:library:Scripts:Libs:graphLib.scpt"

Pour agir sur la fenêtre graphique, graphLib pilote un logiciel graphique scriptable tel que *Photoshop* ou *Illustrator* d'Adobe ou *Smile* de Satimage.fr/software[7].

Actuellement graphLib.scpt existe pour quatre versions utilisant respectivement les applications: "Smile", "Photoshop CS5", "Adobe Photoshop 2018" et "Adobe Photoshop 2021".
Afin de simplifier le passage d'une version à l'autre les accès aux primitives du logiciel graphique ont été réduits et regroupés en seulement cinq routines:

```
openDoc(titre, largeur, hauteur)
fillregion(listeDeSommets, contourMode, colour)
placeText(textString, textPosition, textSize, textColor, textfont)
show(mode)
saveDoc(docRef, filePath)
```

Une fois installée et testée chaque variante peut être optimisée et enrichie à l'aide des primitives spécifiques dont l'application graphique dispose, notamment pour tracer des courbes de Bézier ou créer des dégradés.

Autres modules de bibliothèque

Des variantes de graphLib ont été développées à l'usage didactique sous forme de Libs: TurtleLib et EuclidLib, au lieu d'en faire des applications fermées comme on en trouve sur Internet.

TurtleLib

TurtleLib offre une interface simple, à but pédagogique, inspirée du Logo de Seymour Papert e.a. pour commander les déplacements d'une tortue virtuelle. Elle facilite de manière naturelle les constructions recursives ou fractales de Peano ou Sierpinski et l'interprétation de Systèmes de Lindenmayer.

EuclidLib

EuclidLib s'appuie sur la géométrie plane Euclidienne. Ses routines permettent la construction de figures, limitées à celles construites "à-la-règle-et-au-compas", pour l'observation "expérimentale" de leurs propriétés, telles que l'alignement des points sur la droite d'Euler, sans toutefois en fournir de preuve.

---

[7] *Smile* est gratuit et fonctionne sous Mac OS Mojave, mais pas avec les Mac OS plus récents (64 bits).

textLib

Rappelons encore textLib, une bibliothèque très utile pour la manipulation de texte, tables, nombres, expressions arithmétiques, listes et matrices.

Scripting Additions

Toutes ces bibliothèques ont été conçues en l'absence de "Scripting Additions" et contiennent notamment les définitions de fonctions trigonométriques.
Or, les versions successives de Mac OS imposent parfois ou suppriment ces "Additions".
A noter: leur syntaxe est différente! En effet **abs, sqrt, sin, cos** et **atan** ont le statut d'opérateurs et non de fonctions.
Par exemple "sin(pi/6)" est licite avec graphLib mais doit s'écrire "(**sin** pi/6)" en présence des "Additions".
Dans le doute vérifier les valeurs de "sin(30)-1" , "sin(pi/6)-1" ou "(sin pi/6)-1".

Table des matières