Cette carte détaille les fonctions graphiques disponibles. En annexe, figurent quelques fonctions supplémentaires utilisées dans le cours de Jean-Paul Roy de l'université de Nice-Sophia Antipolis.

**—Images—**

Les images sont des carrés de 200 pixels sur 200. les nombres qui servent de coordonnées pour les images sont compris entre -1 et 1. Le point (0,0) est au centre de l'image.

`image?: Valeur -> bool`
`(image? v)` *reconnaît si v est une image*

`image-vide: -> Image`
`(image-vide)` *rend une image vide*

`image-hauteur: Image -> nat`
`(image-hauteur im)` *rend la hauteur (en pixels) de l'image*

`image-largeur: Image -> nat`
`(image-largeur im)` *rend la largeur (en pixels) de l'image*

`ligne: Nombre^4 -> Image`
`(ligne x1 y1 x2 y2)` *rend une image contenant un segment d'extrémités (x1,y1) et (x2,y2)*

`triangle: Nombre^6 -> Image`
`(triangle x1 y1 x2 y2 x3 y3)` *rend une image contenant un triangle plein de sommets (x1,y1), (x2,y2) et (x3,y3)*

`contour-ellipse: Nombre^4 -> Image`
`(contour-ellipse x1 y1 x2 y2)` *rend une image contenant le tracé d'une ellipse inscrite dans un rectangle de coin inférieur gauche (x1,y1) et supérieur droit (x2,y2)*

`ellipse: Nombre^4 -> Image`
`(ellipse x1 y1 x2 y2)` *rend une image contenant une ellipse pleine inscrite dans un rectangle de coin inférieur gauche (x1,y1) et supérieur droit (x2,y2)*

`superposition: Image * Image * ... -> Image`
`(superposition im1 im2 ...)` *rend l'image obtenue en superposant les images im1, im2, etc. (anciennement nommée* overlay*)*

`quart-de-tour: Image -> Image`
`(quart-de-tour im)` *rend l'image obtenue en la tournant d'un quart de tour dans le sens des aiguilles d'une montre*

`symetrique: Image -> Image`
`(symetrique im)` *rend l'image obtenue par symétrie par rapport à l'axe vertical*

`vignette: Image * Nombre^4 -> Image`
`(vignette im w h x y)` *rend une nouvelle image où im est retaillée aux nouvelles largeur w et hauteur h et positionnée en x y*

**—Pour JP Roy—**

Ces fonctions sont utiles pour le cours de JP Roy :

`empty?: LISTE[alpha] -> bool`
`(empty? L)` *reconnaît si une liste est vide*

`first: LISTE[alpha] -> alpha`
`(first L)` *rend le premier élément de la liste L*
ERREUR *lorsque L est vide*

`rest: LISTE[alpha] -> LISTE[alpha]`
`(rest L)` *rend la liste L privée de son premier élément*
ERREUR *lorsque L est vide*

`second: LISTE[alpha] -> alpha`
`(second L)` *rend le second élément de la liste L*
ERREUR *lorsque L contient moins que deux éléments*

`third: LISTE[alpha] -> alpha`
`(third L)` *rend le troisième élément de la liste L*
ERREUR *lorsque L contient moins que trois éléments*

`fourth: LISTE[alpha] -> alpha`
`(fourth L)` *rend le quatrième élément de la liste L*
ERREUR *lorsque L contient moins que quatre éléments*