Chapitre 2: Images

INF5071 — Infographie

Alexandre Blondin Massé

Université du Québec à Montréal

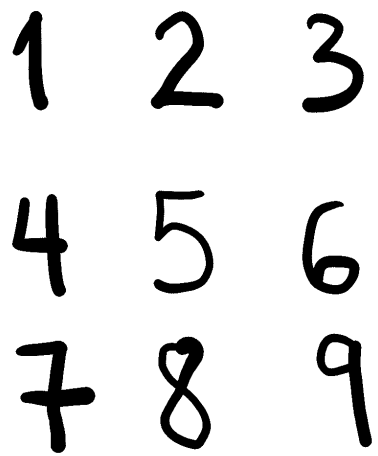
Hiver 2019

Plan

Images et couleurs

Qu'est-ce qu'une image?

- C'est une **représentation visuelle** en 2D de quelque chose
- Généralement de forme **rectangulaire**
- Objet **statique** (non animé)
- Plusieurs **types**: noir et blanc, niveau de gris, couleurs, etc.
- Permet d'encoder de l'**information**



noir et blanc
(*black and white*)

niveau de gris
(*gray scale*)

couleurs
(*colors*)

Deux principaux formats

Vectoriel

- Contenu décrit **mathématiquement**
- Utilise des **primitives géométriques** (lignes, ellipses, arcs, courbes de Bézier, texte, ...)
- La qualité s'adapte aux **transformations** (changement d'échelle, rotations, etc.)
- Doit être transformé sous forme **matricielle** avant le **rendu** (*rendering*)

Matriciel

- Contenu décrit **pixel par pixel**
- Canevas **rectangulaire** (largeur $\times$ hauteur)
- Sensible aux **transformations** (**changements d'échelle**, ...)
- Le rendu se fait généralement **rapidement**

Quelques formats de fichier

- **1987**: GIF = *Graphics Interchange Format*, format **bitmap** compressé sans perte à l'aide de l'algorithme LZW (breveté en 1985), créé par CompuServe
- **1992**: JPEG = *Joint Photographic Experts Group*, format **bitmap** compressé avec perte à l'aide d'une transformée en cosinus discrète, développé par le groupe du même nom
- **1993**: PDF = *Portable Document Format*, format **ouvert** en 2008, développé par Adobe, successeur de PostScript
- **1996**: PNG = *Portable Network Graphics*, format **bitmap ouvert**, développé par le *PNG Development Group*
- **1999**: SVG = *Scalable Vector Graphics*, un format **ouvert** développé par le Word Wide Web Consortium (W3C)
- **2005**: PGF/TikZ = *Portable Graphics Format/TikZ ist kein Zeichenprogramm*, un format **vectoriel** compatible avec TeX/LaTeX, créé par Till Tantau

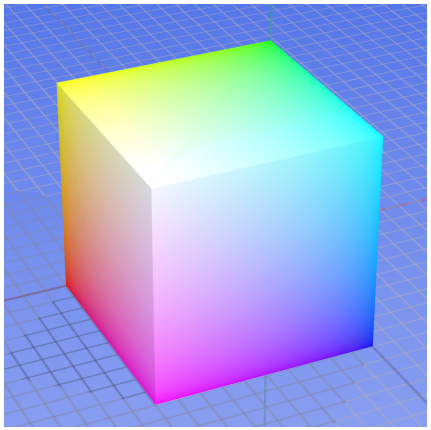
Type de média (*MIME-type*)

Ces formats sont généralement bien **reconnus**:

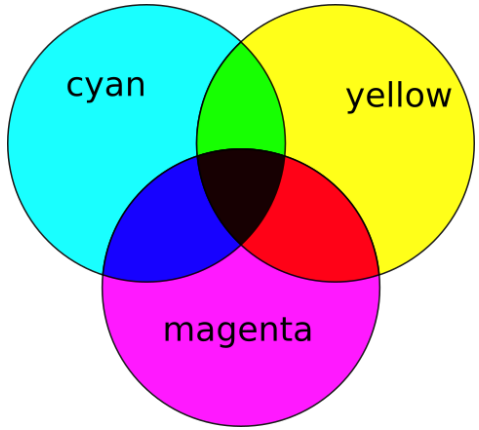
```
$ file *
godot-logo.gif:  GIF image data, version 89a, 138 x 129
godot-logo.jpeg: JPEG image data, JFIF standard 1.01,
    aspect ratio, density 1x1, segment length 16,
    baseline, precision 8, 138x129, frames 3
godot-logo.pdf:  PDF document, version 1.4
godot-logo.png:  PNG image data, 138 x 129,
    8-bit/color RGBA, non-interlaced
godot-logo.svg:  SVG Scalable Vector Graphics image

$ file --mime-type *
godot-logo.gif:  image/gif
godot-logo.jpeg: image/jpeg
godot-logo.pdf:  application/pdf
godot-logo.png:  image/png
godot-logo.svg:  image/svg+xml
```

Systèmes de couleur

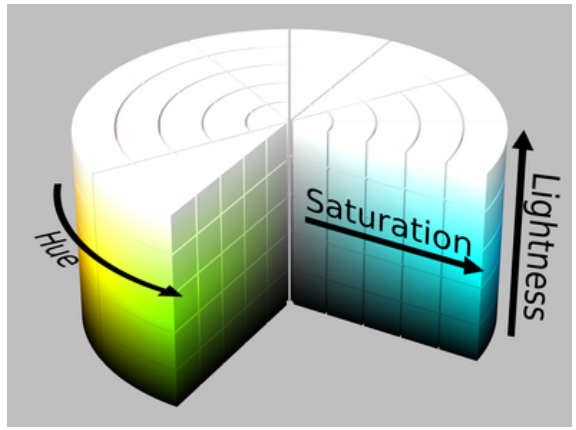
- **RYB** = *Red Yellow Blue*, un système **additif**
- **RGB** = *Red Green Blue*, un système **soustractif**, sans doute le plus utilisé
- **CMYK** = *Cyan Magenta Yellow blacK*, un système **additif**, utilisé surtout avec les imprimantes
- **HSI/HSV/HSL** = *Hue Saturation Intensity/Value/Lightness*, des modèles qui semblent plus intuitifs pour les humains



espace RGB
(cube)
Source: Wikipedia



système CMYK
(diagramme de Venn)



espace HSV
(cylindre)
Source: Wikipedia

Systèmes RGB et HSI

Représentation

- Dans le système **RGB**, une couleur est un **triplet** (r, g, b)
- De la même façon, dans le système **HSI**, c'est un triplet (h, s, i)
- En théorie, chaque **composante** est entre 0 et 1
- En pratique, entre 0 et $2^d - 1$, où d est appelé **profondeur de couleur**

Changement de variables

$$
\begin{aligned}
i &= \frac{r+g+b}{3} \\
s &= 1 - \frac{3\min\{r,g,b\}}{r+g+b} \\
h &= \arccos\left(\frac{(r-g)+(r-b)}{2\sqrt{(r-g)^2+(r-b)(g-b)}}\right)
\end{aligned}
$$

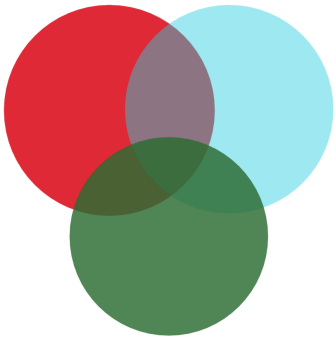
Transparence

- La plupart des modèles de couleur proposent un **4e canal**
- Appelé canal **alpha**, souvent représenté par la **variable** a

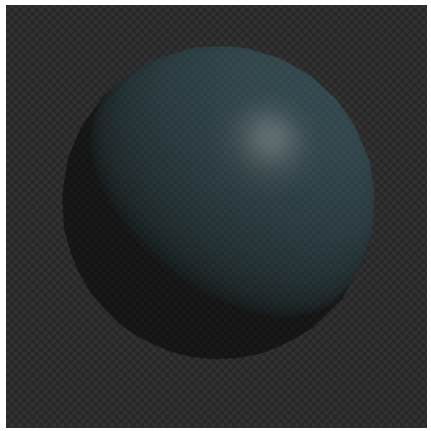
$\rightarrow$ $a = 1$: la couleur est **opaque**

$\rightarrow$ $a = 0$: la couleur est **transparente**

- L'acronyme **RGBA** permet de préciser la présence de ce 4e canal
- Utilise pour **superposer/combiner** plusieurs images
- Ou encore pour contrôler leur **visibilité**
- Souvent, l'**arrière-plan** est transparent, identifié par un **damier**



disques transparents



arrière-plan transparent

Considérations artistiques

Théorie de la couleur

- **« Psychologie »** des couleurs (chaudes, froides, dynamiques, ...)
- Couleurs **complémentaires**
- Différentes **théories** sur la meilleure façon d'arranger des couleurs
- **Schémas/thèmes** de couleur (*color schemes*)
- Varie d'une **culture** à une autre

À éviter/à utiliser avec parcimonie

- Plusieurs couleurs de **teintes différentes**
- Les couleurs ayant des **teintes proches**
- Les couleurs **saturées**

Géométrie 2D

Points et vecteurs

- **Plan euclidien**: c'est l'*ensemble* $\mathbb{R}^2 = \mathbb{R} \times \mathbb{R}$
- **Point**: un *élément* $P = (P_x, P_y)$ de $\mathbb{R}^2$
- **Vecteur**: un *élément* $\vec{u} = (u_1, u_2)$ de $\mathbb{R}^2$
- **Norme**: si $\vec{u}$ est un vecteur, alors $\|\vec{u}\| = \sqrt{u_1^2 + u_2^2}$

Opérations

Soient $P = (P_x, P_y)$, $Q = (Q_x, Q_y)$ des **points**, $\vec{u} = (u_1, u_2)$, $\vec{v} = (v_1, v_2)$ des **vecteurs** et $k \in \mathbb{R}$

- **Différence** de points: $Q - P = \overrightarrow{PQ} = (Q_x - P_x, Q_y - P_y)$
- **Somme/différence** de vecteurs: $\vec{u} \pm \vec{v} = (u_1 \pm v_1, u_2 \pm v_2)$
- **Point plus vecteur**: $P \pm \vec{u} = (P_x \pm u_1, P_y \pm u_2)$
- **Multiplication** par un scalaire: $k\vec{u} = (ku_1, ku_2)$
- **Produit scalaire** entre deux vecteurs: $\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2$

Note: $P + Q$ n'est **pas définie**!!

Trois transformations importantes

Il est pratique de représenter un point par un **vecteur colonne**:

$$P = \begin{bmatrix} x \\ y \end{bmatrix}$$

- **Translation** de vecteur $v \in \mathbb{R}^2$:

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} x \\ y \end{bmatrix} + \vec{v}$$

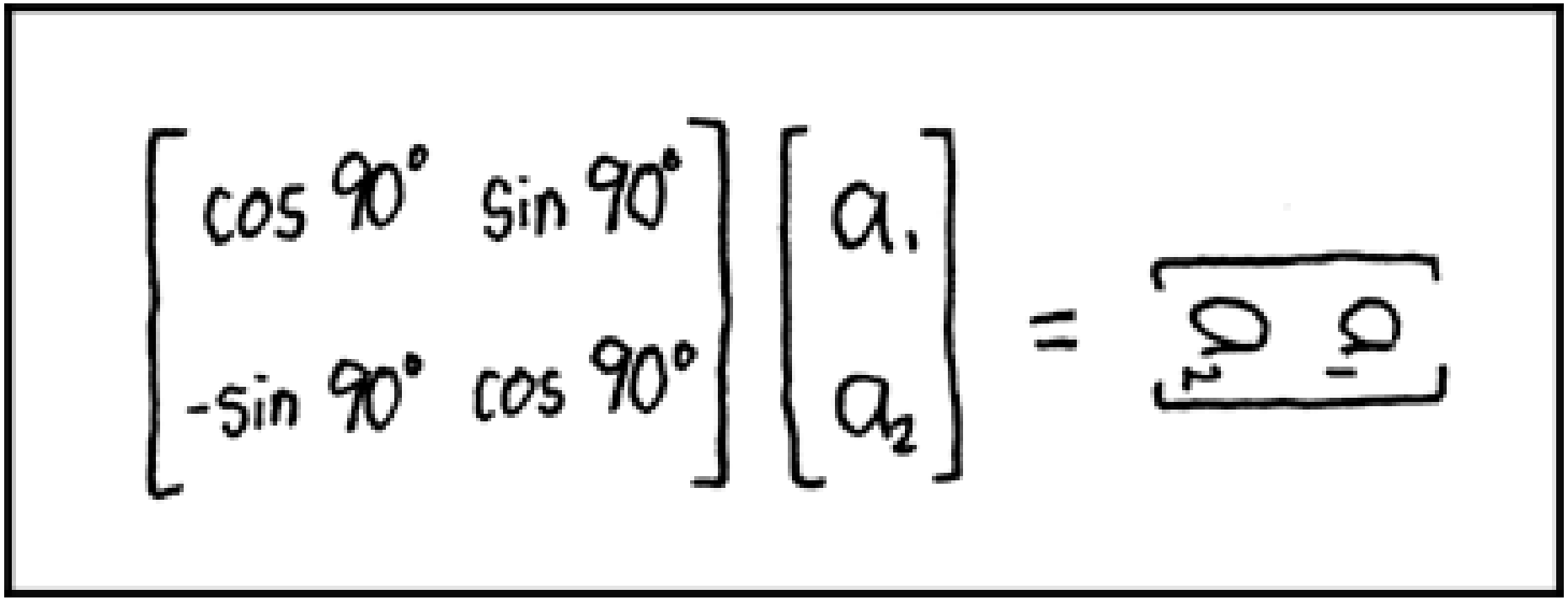
- **Rotation** d'angle θ en sens **anti-horaire** autour de l'origine:

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- **Changement d'échelle** de facteurs $a \in \mathbb{R}$ et $b \in \mathbb{R}$:

$$\begin{bmatrix} x \\ y \end{bmatrix} \mapsto \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Rotations 2D



Source: https://xkcd.com/184/

Lieux géométriques

- **Lieu géométrique**: un *sous-ensemble* de $\mathbb{R}^2$
- **Dimension**: $\approx$ degré de liberté

$\rightarrow$ **0D**: longueur $= 0$

$\rightarrow$ **1D**: longueur $\neq 0$, aire $= 0$

$\rightarrow$ **2D**: aire $\neq 0$

Exemples de lieux géométriques

- **0D**: point, ensemble de points isolés
- **1D**: segment, droite, demi-droite, cercle, ellipse, parabole, ...
- **2D**: disque, rectangle plein, demi-plan, plan, ...

Décrire un lieu géométrique

- **En mots**: le segment de droite reliant les points $(10, 30)$ et $(60, -20)$
- À l'aide d'une **(in)équation cartésienne**: le cercle de rayon 6 centré en $(20, -40)$ est défini par l'équation

$$(x - 20)^2 + (y + 40)^2 = 36$$

- À l'aide d'une **(in)équation vectorielle**: si $\vec{v}$ est un vecteur non nul et P_0 un point, alors l'unique droite Δ passant par P_0 dans la direction de $\vec{v}$ est donné par l'ensemble

$$\Delta = \{P \in \mathbb{R}^2 \mid P = P_0 + t\vec{v}, \quad t \in \mathbb{R}\}$$

Lieux géométriques linéaires

Soit P_0 un point et $\vec{v}$ un vecteur non nul

- **Droite**: l'unique droite Δ qui passe par P_0 dans la direction de $\vec{v}$ est

$$\Delta = \{P \in \mathbb{R}^2 \mid P = P_0 + t\vec{v}, \quad t \in \mathbb{R}\}$$

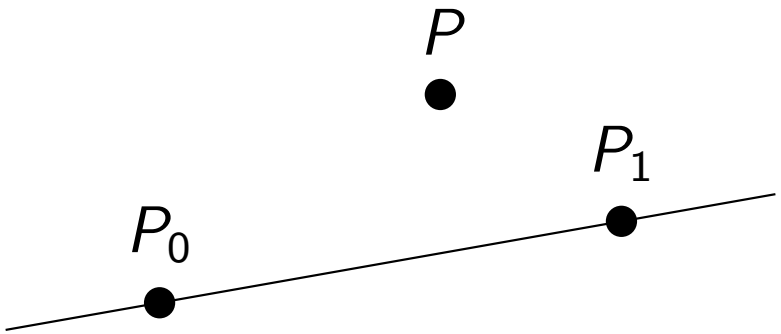
- **Demi-droite**: l'unique demi-droite H qui démarre en P_0 dans la direction de $\vec{v}$ est

$$H = \{P \in \mathbb{R}^2 \mid P = P_0 + t\vec{v}, \quad t \in \mathbb{R}_+\}$$

- **Segment**: si P_1 est un point distinct de P_0, alors l'unique segment qui relie P_0 et P_1 est

$$S = \{P \in \mathbb{R}^2 \mid P = P_0 + (P_1 - P_0)t, \quad t \in [0, 1]\}$$

Question sur les lieux géométriques linéaires (1/3)

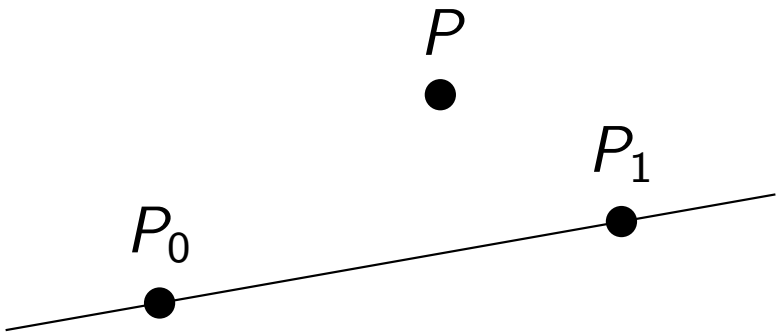


- Soient $P_0 = (x_0, y_0)$, $P_1 = (x_1, y_1)$ et $P = (x, y)$
- Comment **décide-t-on** si P appartient à la **droite** ou le **segment** qui passent par P_0 et P_1?
- Solution **cartésienne**: on cherche s'il existe t telle que

$$(x, y) = (x_0, y_0) + (x_1 - x_0, y_1 - y_0)t$$

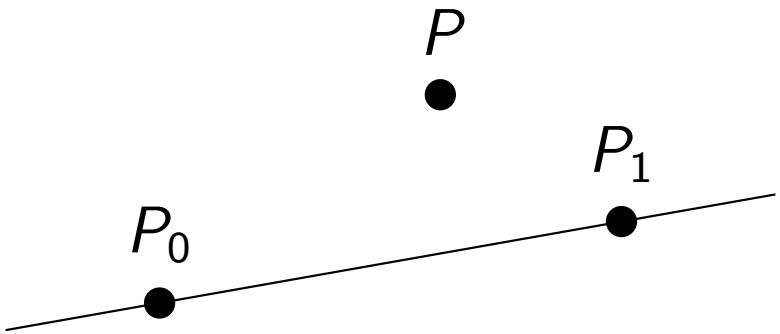
- **Question** avec $P_0 = (1, 3)$, $P_1 = (5, -5)$ et $P = (3, 6)$?

Question sur les lieux géométriques linéaires (2/3)



- D'un point de vue **informatique**, il y a mieux
- Il suffit de réfléchir de façon **vectorielle**
- Quelle devrait être la **relation** entre $\overrightarrow{P_0P}$ et $\overrightarrow{P_0P_1}$?

Question sur les lieux géométriques linéaires (2/3)

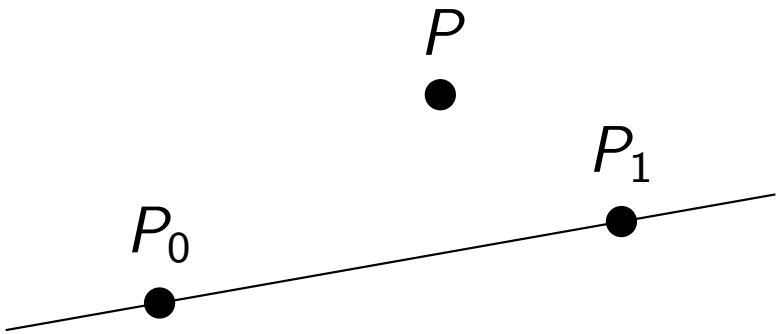


- D'un point de vue **informatique**, il y a mieux
- Il suffit de réfléchir de façon **vectorielle**
- Quelle devrait être la **relation** entre $\overrightarrow{P_0P}$ et $\overrightarrow{P_0P_1}$?

$\rightarrow$ ils doivent être **colinéaires**!

- **Opération** sur les vecteurs qui permet de vérifier ça?

Question sur les lieux géométriques linéaires (2/3)



- D'un point de vue **informatique**, il y a mieux
- Il suffit de réfléchir de façon **vectorielle**
- Quelle devrait être la **relation** entre $\overrightarrow{P_0P}$ et $\overrightarrow{P_0P_1}$?

$\rightarrow$ ils doivent être **colinéaires**!

- **Opération** sur les vecteurs qui permet de vérifier ça?

$\rightarrow$ le produit **vectoriel**!

Produit vectoriel

Définition

- Soient $\vec{u} = (u_1, u_2, u_3)$ et $\vec{v} = (v_1, v_2, v_3)$
- Alors le **produit vectoriel** de $\vec{u}$ et $\vec{v}$ est

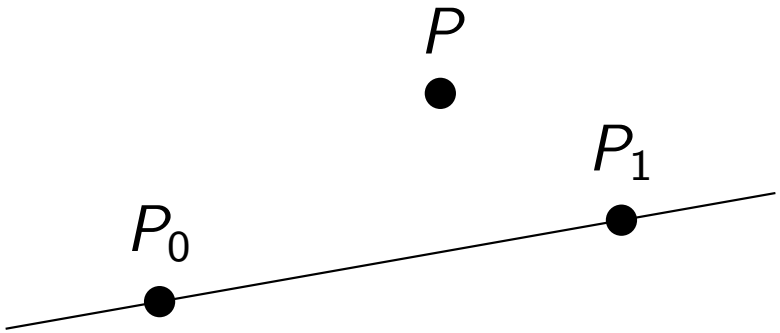
$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \end{vmatrix} = (u_2 v_3 - u_3 v_2, u_3 v_1 - u_1 v_3, u_1 v_2 - u_2 v_1)$$

- Si les vecteurs sont en **2D**, on met leur **3e composante** à **0**

Propriété

Les vecteurs $\vec{u}$ et $\vec{v}$ sont **colinéaires** si et seulement si $\vec{u} \times \vec{v} = \vec{0}$

Question sur les lieux géométriques linéaires (3/3)

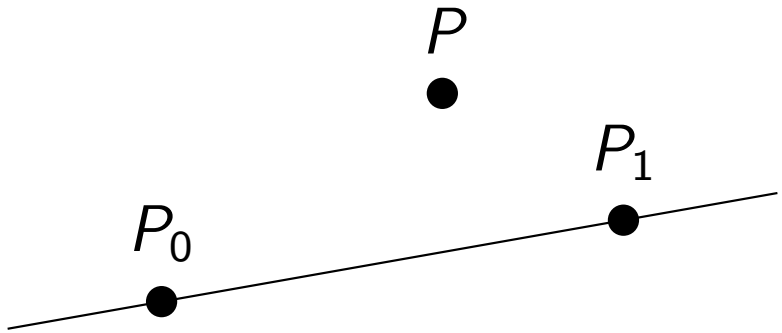


- On a donc que P **appartient à la droite** si et seulement si

$$\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x - x_0 & y - y_0 & 0 \\ x_1 - x_0 & y_1 - y_0 & 0 \end{vmatrix} = (0, 0, 0)$$

  si et seulement si $(x - x_0)(y_1 - y_0) = (y - y_0)(x_1 - x_0)$
- **Segment**?

Question sur les lieux géométriques linéaires (3/3)



- On a donc que P **appartient à la droite** si et seulement si

$$\begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ x - x_0 & y - y_0 & 0 \\ x_1 - x_0 & y_1 - y_0 & 0 \end{vmatrix} = (0, 0, 0)$$

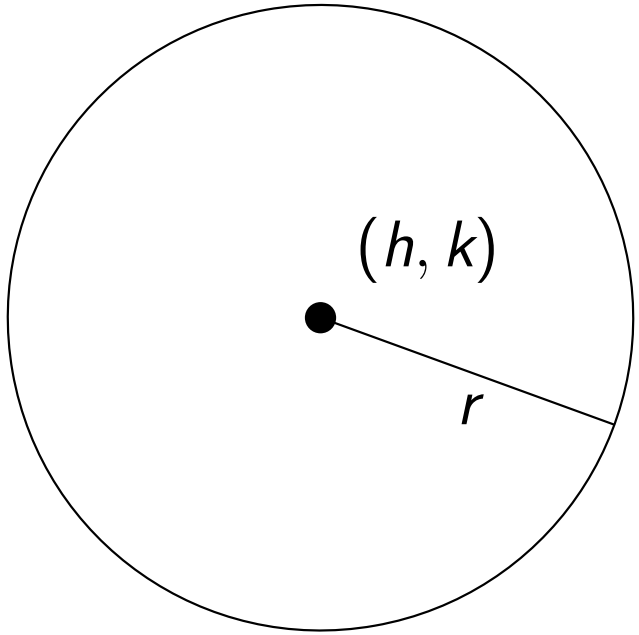
  si et seulement si $(x - x_0)(y_1 - y_0) = (y - y_0)(x_1 - x_0)$
- **Segment**?

$\rightarrow$ On vérifie d'abord si P est **sur la droite**

$\rightarrow$ Puis on calcule $t = (x - x_0)/(x_1 - x_0)$ ou $t = (y - y_0)/(y_1 - y_0)$

$\rightarrow$ On se **trouve sur le segment** si et seulement si $t \in [0, 1]$

Cercles



- Soit $O = (h, k)$ un point et $r \geq 0$ un réel
- Alors il existe un **unique cercle** C de rayon r centré en O
- **Équation cartésienne**: $(x - h)^2 + (y - k)^2 = r^2$

Question sur les cercles

- Soit C un cercle de centre O et de rayon r
- Soit P un point quelconque
- Comment décide-t-on si P est **sur** le cercle? à l'**intérieur** du cercle? à l'**extérieur** du cercle?

Question sur les cercles

- Soit C un cercle de centre O et de rayon r
- Soit P un point quelconque
- Comment décide-t-on si P est **sur** le cercle? à l'**intérieur** du cercle? à l'**extérieur** du cercle?

$\rightarrow$ Plus pratique d'avoir une **définition vectorielle**:

$$C = \left\{ P \in \mathbb{R}^2 \;\middle|\; \left\| \overrightarrow{OP} \right\| = r \right\}$$

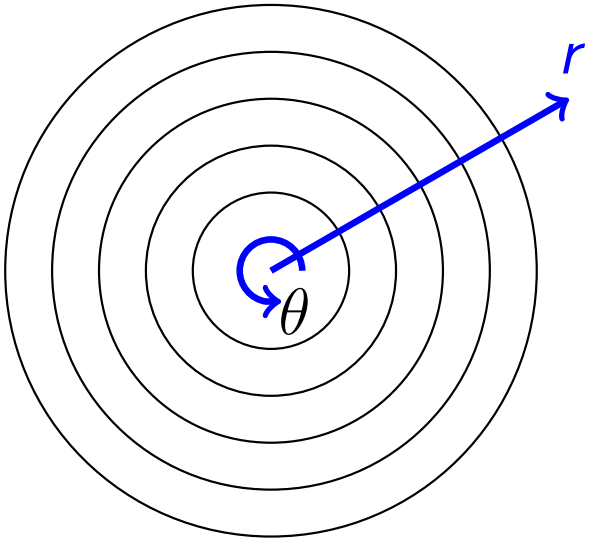
$\rightarrow$ On calcule $\left\| \overrightarrow{OP} \right\|^2 = \overrightarrow{OP} \cdot \overrightarrow{OP}$

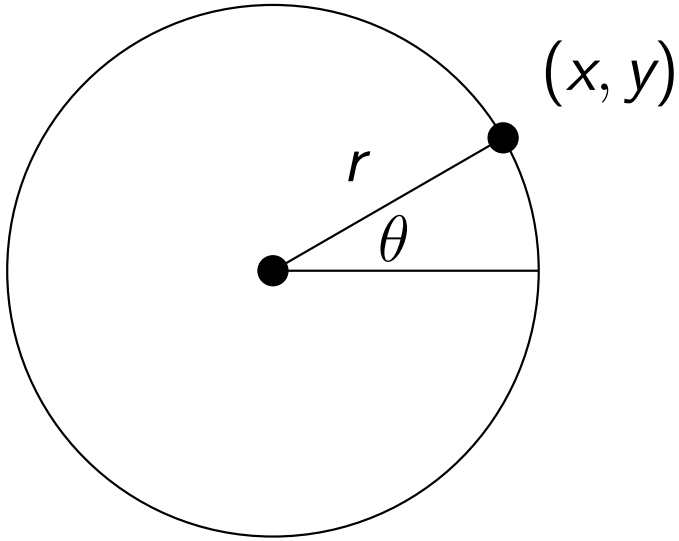
$\rightarrow$ Si $\left\| \overrightarrow{OP} \right\|^2 < r^2$, alors on est à l'**intérieur**

$\rightarrow$ Si $\left\| \overrightarrow{OP} \right\|^2 = r^2$, alors on est **sur** le cercle

$\rightarrow$ Si $\left\| \overrightarrow{OP} \right\|^2 > r^2$, alors on est à l'**extérieur**

Coordonnées polaires (1/2)





- Certains **lieux géométriques** ont une structure liée à des **cercles**
- Il est donc plus facile de les décrire en **coordonnées polaires**
- **Cartésiennes**: abscisse x et ordonnée y
- **Polaires**: distance r et angle θ

Coordonnées polaires (2/2)

- Soit P un point
- On écrit $P = (x, y)$ en coordonnées **cartésiennes**
- Et $P = (r : \theta)$ en coordonnées **polaires**
- Les relations entre x, y, r et θ sont décrites par les formules suivantes:

$$
\begin{aligned}
x &= r\cos\theta \\
y &= r\sin\theta \\
r &= \sqrt{x^2 + y^2} \\
\theta &= \arctan\left(\frac{y}{x}\right)
\end{aligned}
$$

Exemples sur les coordonnées polaires

- Le **cercle** C de centre $(0, 0)$ et de rayon r devient simplement

$$C = \{(r : \theta) \mid 0 \leq \theta < 2\pi\}$$

- L'**arc de cercle** A autour de $(0, 0)$, de rayon r et entre les angles θ_1 et θ_2 est donc

$$A = \{(r : \theta) \mid \theta_1 \leq \theta \leq \theta_2\}$$

- La **demi-droite** D partant de $(0, 0)$ en direction du vecteur non nul $\vec{u} = (u_1, u_2)$?

Conique

- **Courbe** qu'on peut obtenir par **section** d'un cône
- Équation **cartésienne** polynomiale de degré au plus 2

Trois coniques non dégénérées

- **Ellipses** (cas particulier du cercle)
- **Paraboles**
- **Hyperboles**

Cas dégénérés

- Point
- Segment
- Demi-droite
- Droite

Courbe générale

Fonction vectorielle

- C'est une fonction qui retourne un **vecteur**
- On ajoute souvent une **flèche** sur la fonction pour insister

Courbe paramétrée

- Soit C une courbe continue
- Alors il existe

$\rightarrow$ deux **réels** a et b, avec $a < b$

$\rightarrow$ deux **fonctions réelles** $x, y : [a, b] \rightarrow \mathbb{R}$

$\rightarrow$ et une **fonction vectorielle**

$$\begin{aligned} \vec{r} : [a, b] &\rightarrow \mathbb{R}^2 \\ t &\mapsto (x(t), y(t)) \end{aligned}$$

dont C est l'**image**, c'est-à-dire que $C = \vec{r}([a, b])$

Exemples de courbes paramétrées

- Le **segment** qui relie deux points P_0 et P_1 est l'image de

$$\vec{r}(t) = P_0 + (P_1 - P_0)t, \quad 0 \leq t \leq 1$$

- Le **cercle** de rayon r centré à l'origine est l'image de

$$\vec{r}(t) = r(\cos t, \sin t), \quad 0 \leq t \leq 2\pi$$

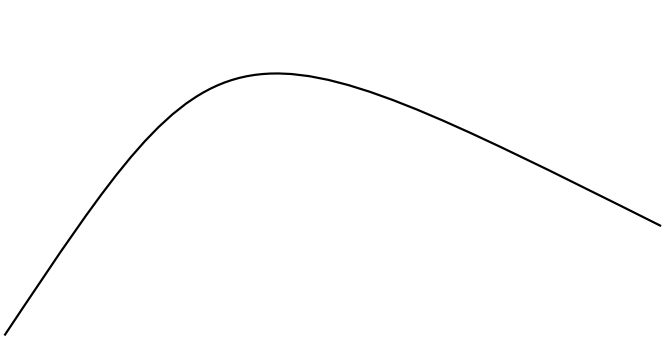
- Le **rectangle** de coins $(0,0)$, $(100,0)$, $(100,200)$ et $(0,200)$:

$$\vec{r}(t) = \begin{cases} (100t, 0) & \text{si } 0 \leq t \leq 1 \\ (100, 200(t-1)) & \text{si } 1 \leq t \leq 2 \\ (100(3-t), 200) & \text{si } 2 \leq t \leq 3 \\ (0, 200(4-t)) & \text{si } 3 \leq t \leq 4 \end{cases}$$

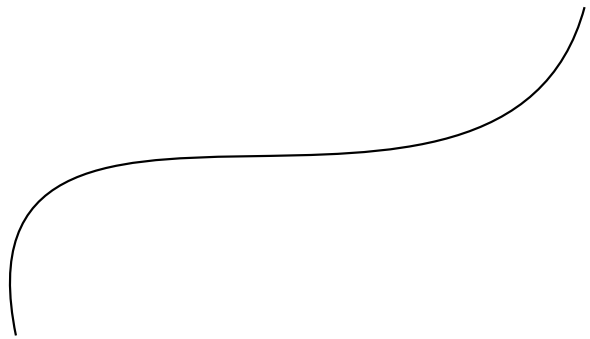
- L'**ellipse** de rayons a, b, centrée en (h,k), est l'image de

$$\vec{r}(t) = (h,k) + (a\cos t, b\sin t), \quad 0 \leq t \leq 2\pi$$

Courbes de Bézier



quadratique



cubique

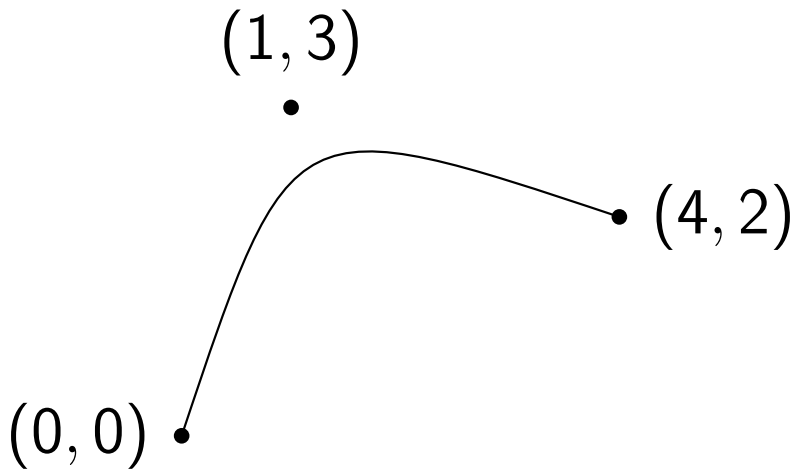
- **Courbes** très utilisées en infographie
- Elles sont décrites par des **polynômes** de degré d
- Segment = courbe de Bézier où le degré du polynôme est $d = 1$
- En **pratique**, on utilise surtout les courbes de Bézier **quadratiques** ($d = 2$) et **cubiques** ($d = 3$)
- **Représentation**: à l'aide de $d + 1$ points de **contrôle**

Courbe de Bézier quadratique

- **Trois** points de contrôle P_0, P_1 et P_2
- L'**ordre** d'énumération est important
- **Débute** en P_0 et **termine** en P_2;
- On **ne passe pas** par le point P_1, mais on s'en **approche**
- **Paramétrisation**:

$$\vec{r}(t) = \vec{P_0}(1-t)^2 + 2\vec{P_1}t(1-t) + \vec{P_2}t^2, \quad 0 \leq t \leq 1$$

- **Exemple** avec $P_0 = (0,0)$, $P_1 = (1,3)$ et $P_2 = (4,2)$:

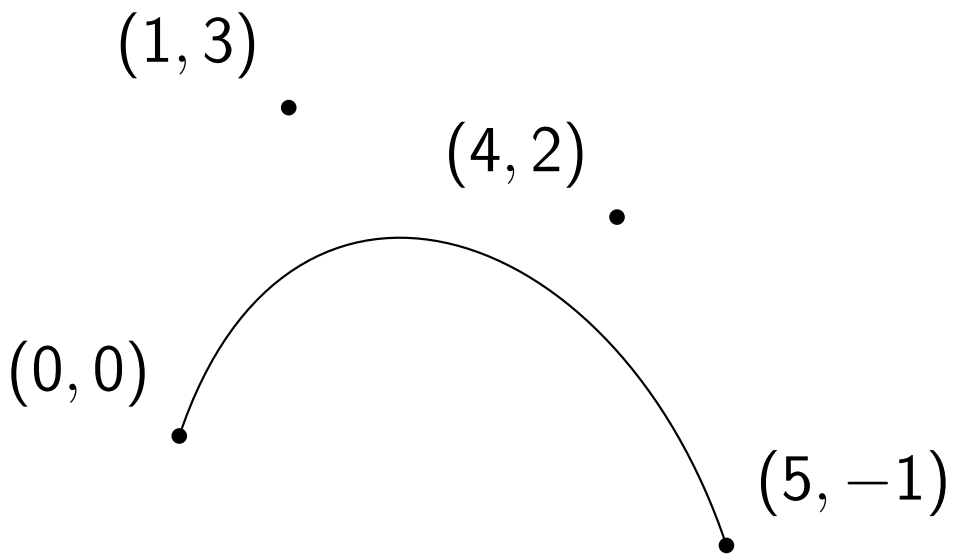


Courbe de Bézier cubique

- **Quatre** points de contrôle P_0, P_1, P_2 et P_3
- **Débute** en P_0 et **termine** en P_3;
- **Paramétrisation**: pour $0 \leq t \leq 1$,

$$\vec{r}(t) = \vec{P_0}(1-t)^3 + 3\vec{P_1}t(1-t)^2 + 3\vec{P_2}t^2(1-t) + \vec{P_3}t^3$$

- **Exemple** avec $P_0 = (0,0)$, $P_1 = (1,3)$, $P_2 = (4,2)$ et $P_3 = (5,-1)$:



Générer des images

Trois stratégies

À la main

- En utilisant un logiciel de **dessin**
- Ou en apportant des **correctifs manuels** à une autre image

De façon semi-automatique

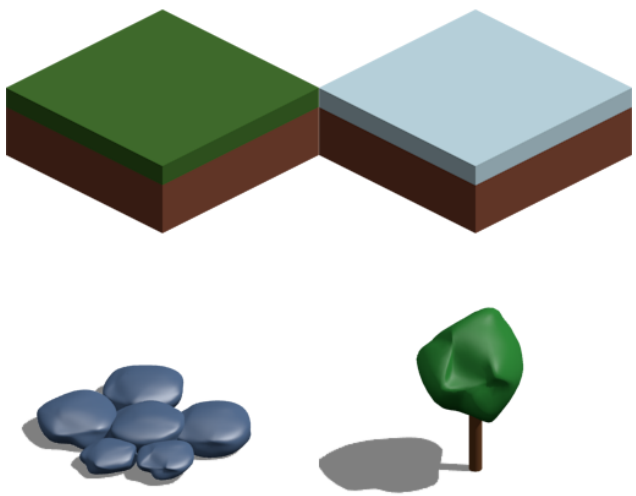
- En **décrivant** mathématiquement le dessin (SVG, PGF/TikZ)
- En **scriptant** à l'aide de logiciels (Krita, Gimp, Inkscape, Blender, ...)

Par programmation

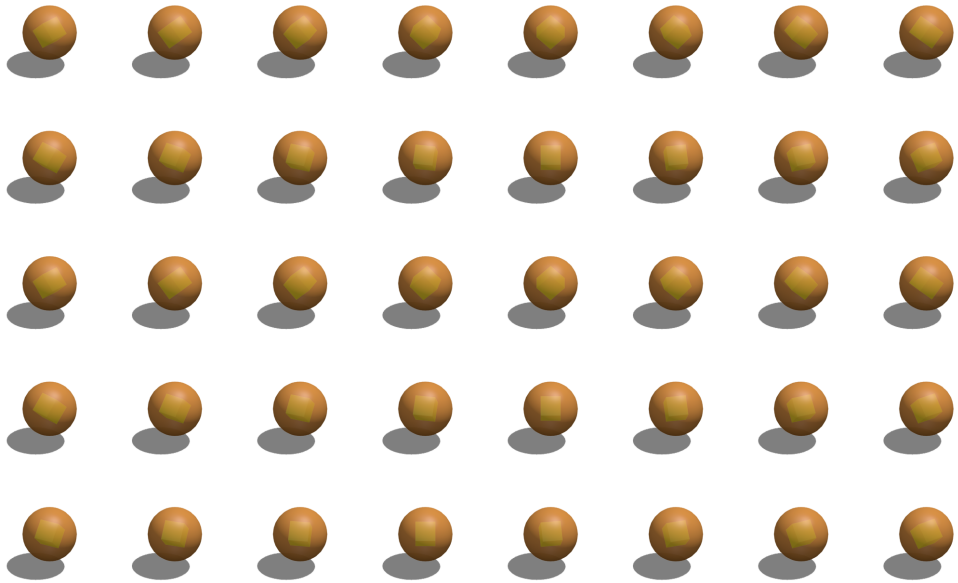
- En utilisant des **applications** (exemple ImageMagick)
- En utilisant une **bibliothèque** graphique (exemple Cairo, Pillow)
- En appliquant des **filtres** (exemple G'MIC)

Feuilles de sprites

- C'est simplement une **image**
- Qui permet de rassembler et d'organiser **plusieurs images**
- Généralement de façon **logique**
- Très utile pour les jeux basés sur des **tuiles**



ensemble de tuiles



sprite animée

Création de feuilles de sprites

Plusieurs façons

- À partir d'**images vectorielles**, facilite l'organisation logique
- Ou d'**images matricielles** en utilisant des grilles et des couches
- On peut ensuite les **agencer** à l'aide d'un éditeur de **cartes** (Tiled par exemple)

Démonstration: exemple avec Blender

- Permet de facilement générer des tuiles **isométriques**
- Autant pour générer des tuiles **statiques**...
- ...que des tuiles **animées**
- Voir Isometric Tiles in Blender de C. Bellanger

Le format SVG

Qu'est-ce que c'est?

- SVG = *Scalable Vector Graphics*
- Format **vectoriel ouvert**
- Développé depuis **1999**
- Par le Word Wide Web Consortium (W3C)
- Supporté par la plupart des **navigateurs**
- Supporte les **filtres**
- Supporte les **animations**
- Utilisé par Inkscape pour la sauvegarde

Références

- Tutoriel introductif
- Documentation officielle

Primitives géométriques

- **segment** (*line*) d'un point (x_1, y_1) à un point (x_2, y_2)
- **rectangle** de coin (x, y) et de dimensions $w \times h$
- **cercle** de centre (h, k) et de rayon r
- **ellipse** de centre (h, k) et de rayons a, b
- **polysegment** (*polyline*) une suite de points $((x_i, y_i))_{i=1,\ldots,n}$

$\rightarrow$ **fermé**: la courbe revient à son point de départ

$\rightarrow$ **ouvert**: dans le cas contraire

- **texte** de coin (x, y), avec une police de **taille** spécifique

Remarques

- Le **coin** est souvent celui en *haut*, à *gauche*
- On peut contrôler la couleur du **trait**, de **remplissage**
- De façon plus générale, on peut modifier le **trait** (*stroke*)

Exemple: primitives géométriques


```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="1000">
  <rect x="10" y="10" width="180" height="180"
        stroke="blue" stroke-width="3" fill="#FFFF00"/>

  <line x1="210" y1="10" x2="390" y2="180" stroke="blue" stroke-width="5"/>

  <rect x="410" y="10" width="180" height="180"
        stroke="orange" stroke-width="3" fill="#FFFFFF"/>
  <text x="420" y="130" font-size="80" fill="orange">SVG</text>

  <polyline fill="red" stroke="blue" stroke-width="3"
        points="610,100 700,10 790,100 700,190 610,100"/>

  <circle  cx="850" cy="50"  r="20" stroke="blue" fill="cyan"/>
  <circle  cx="850" cy="150" r="30" stroke="red"  fill="orange"/>
  <ellipse cx="950" cy="100" rx="40" ry="70" stroke="black" fill="white"/>
</svg>
```

Chemins

- La notion de **chemin** (*path*) est une généralisation des primitives géométriques
- Disponible dans tout format **vectoriel**
- Décrit essentiellement un **tracé**
- Différents types de **sous-chemins**:

→ **déplacement** (*moveto*): sans tracer
→ **ligne** (*lineto*): segment
→ **arc**: arc de cercle ou d'ellipse
→ **courbe de Bézier**: quadratique ou cubique
→ **fermeture**: pour boucler le chemin

- Prend aussi en compte des **attributs** supplémentaires:

→ **style du trait**: couleur, style, marqueur, terminaisons, jointures, opacité
→ **remplissage**: couleur, gradient, motif

Sous-chemins dans le format SVG

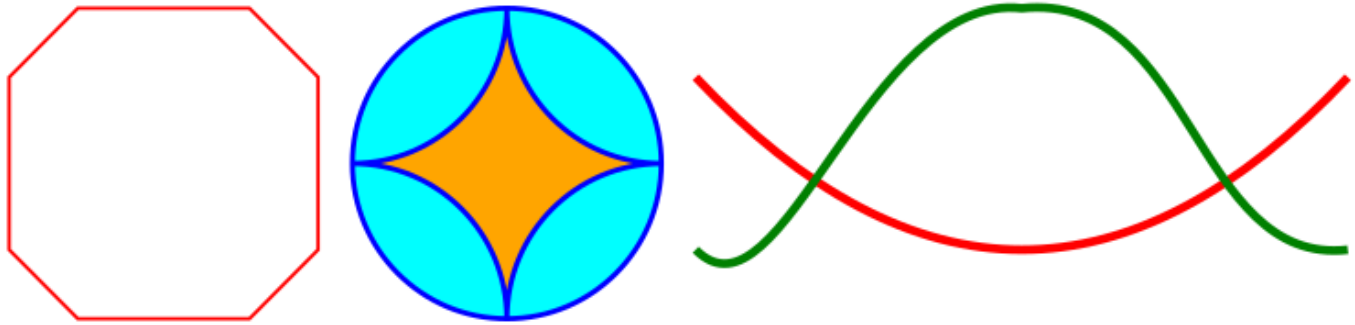
- `M x y`: se **déplacer** sans trait à `(x,y)`
- `L x y`: tracer un **segment** jusqu'à `(x,y)`
- `A rx ry a f1 f2 x y`: tracer un **arc elliptique** jusqu'à `(x,y)`, l'ellipse ayant des rayons `(rx,ry)`, inclinée selon l'angle `a` et

$$\texttt{f1} = \begin{cases} 0, & \text{pour l'arc court;} \\ 1, & \text{pour l'arc long,} \end{cases} \qquad \texttt{f2} = \begin{cases} 0, & \text{pour le sens anti-horaire;} \\ 1, & \text{pour le sens horaire.} \end{cases}$$

- `Q x1 y1 x2 y2`: tracer une courbe de Bézier **quadratique** jusqu'à `(x2,y2)` avec point de contrôle `(x1,y1)`
- `C x1 y1 x2 y2 x3 y3`: tracer une courbe de Bézier **cubique** jusqu'à `(x3,y3)` avec points de contrôle `(x1,y1)` et `(x2,y2)`
- `Z` ou `z` pour refermer un chemin

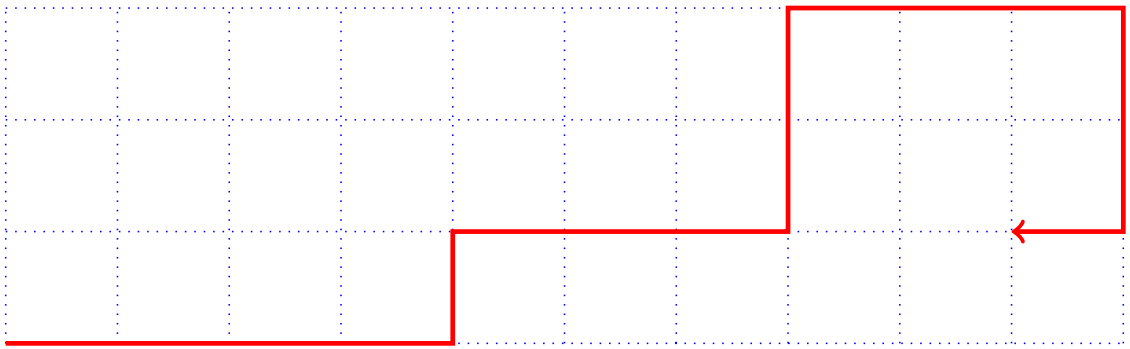
Note: les lettres `M`, `L`, `A`, `Q`, `C` peuvent être remplacées par `m`, `l`, `a`, `q`, `c` pour que les coordonnées soient **relatives** au point courant

Exemple: chemins



```
<svg xmlns="http://www.w3.org/2000/svg" height="200" width="800">
  <path d="M 10 50 l 0 100 l 40 40 l 100 0 l 40 -40
           l 0 -100 l -40 -40 l -100 0 z"
        fill="white" stroke="red" stroke-width="2"/>

  <circle cx="300" cy="100" r="90" fill="cyan"
          stroke="blue" stroke-width="3"/>
  <path d="M 210 100
           a 90 90 0 0 0 90 -90 a 90 90 0 0 0 90 90
           a 90 90 0 0 0 -90 90 a 90 90 0 0 0 -90 -90
           z" stroke="blue" fill="orange" stroke-width="3"/>

  <path d="M 410 50 q 190 200 380 0"
        stroke="red" stroke-width="5" fill="none"/>
  <path d="M 410 150
           c 50 50 100 -150 190 -140
           c 100 -10 100 150 190 140
           " stroke="green" stroke-width="5" fill="none"/>
</svg>
```

PGF/TikZ



```
\begin{tikzpicture}
  \draw[blue, dotted] (0,0) grid (10,3);
  \draw[red, very thick, ->] (0,0) -- ++ (4,0) --
    ++ (0,1) -- ++ (3,0) -- ++ (0,2) -- ++ (3,0) --
    ++ (0,-2) -- ++ (-1,0);
\end{tikzpicture}
```

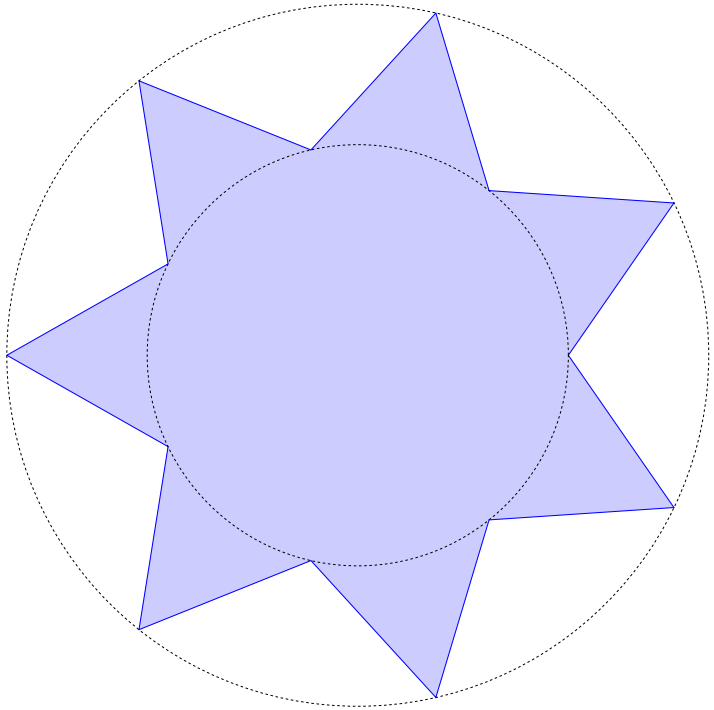
- PGF = *Portable Graphics*
- TikZ = *TikZ ist kein Zeichenprogramm*
- Documentation officielle: PGF/TikZ
- Développé depuis **2005**
- Créé par **Till Tantau**, le créateur de Beamer
- Aussi développé par **Christian Feuersänger**

Exemple: primitives géométriques


```
\begin{tikzpicture}
  \draw[draw=blue, fill=yellow, thick] (0.1,0.1) rectangle (1.9,1.9);

  \draw[draw=blue, very thick] (2.1,1.9) -- ++ (1.8,-1.8);

  \draw[draw=orange, thick] (4.1,0.1) rectangle (5.9,1.9);
  \node at (5,1) {\huge\color{orange}TikZ};

  \draw[draw=blue, fill=red, thick] (6.1,1) -- (7,0.1)
    -- (7.9,1) -- (7,1.9) -- (6.1,1);

  \draw[draw=blue,  fill=cyan]   (8.5,1.5) circle  (0.2cm);
  \draw[draw=red,   fill=orange] (8.5,0.5) circle  (0.3cm);
  \draw[draw=black, fill=white]  (9.5,1.0) ellipse (0.4cm and 0.7cm);
\end{tikzpicture}
```

Exemple: dessiner une étoile

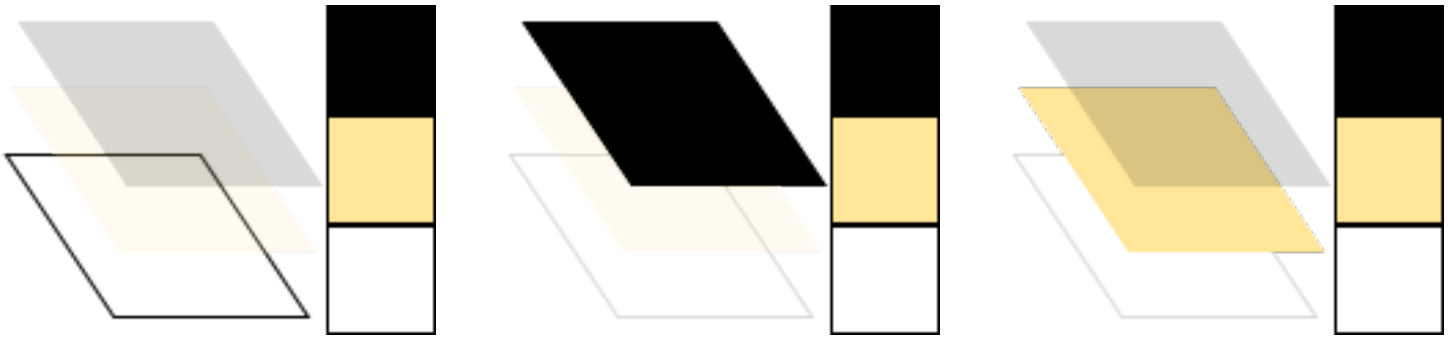
```
\usetikzlibrary{math}

\begin{tikzpicture}
  % Variables
  \newdimen\r1
  \newdimen\r2
  \tikzmath{
    \n = 7;         % Nombre de pointes
    \r1 = 0.3cm;    % Rayon intérieur
    \r2 = 0.5cm;    % Rayon extérieur
    \a = 180 / \n; % Angle entre pointes
 }
  % On trace l'étoile
  \draw[draw=blue, very thick, fill=blue!20]
    (0:\r1) \foreach \i in {1,2,...,\n}  {
    -- (2*\a*\i-\a:\r2) -- (2*\a*\i:\r1) };
  % Puis les cercles pour visualiser
  \draw[dashed] (0,0) circle (\r1);
  \draw[dashed] (0,0) circle (\r2);
\end{tikzpicture}
```



- Noter l'utilisation des **coordonnées polaires**
- Simplifie grandement la **description**
- On peut **scripter** (boucle `foreach`) dans les commandes

Cairo

- **Site officiel:** https://www.cairographics.org/
- **Licence**: LGPL (*Lesser General Public License*) ou MPL (*Mozilla Public License*)
- En **C**, mais utilisable dans la plupart des langages
- Modèle en **3 couches**



destination source masque

Source: Cairo

Polices de caractères

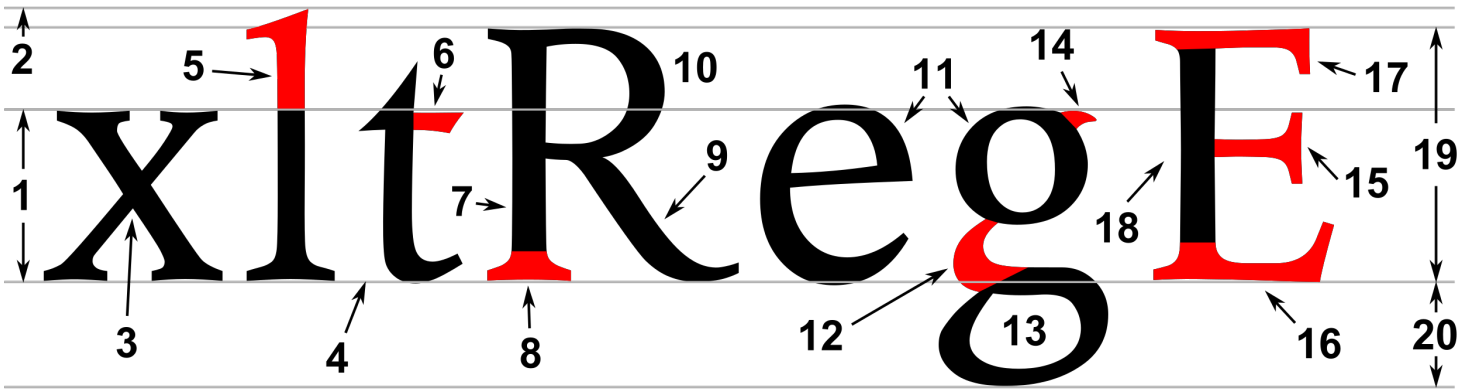
Généralités

- **Glyphe** (*glyph*): représentation graphique d'un caractère ou d'un accent
- **Fonte de caractères** (*fonts*) = ensemble de glyphes
- **Police de caractère** = ensemble de fontes de caractères
- **Empattement**: avec ou sans (serif)
- **Corps**: tailles
- **Graisse**: gras, épaisseur
- **Style:** romain, italique, ombré, décoratif

Considérations artistiques

- Prioriser l'harmonie **entre lettres** plutôt que chaque lettre
- Uniformiser les **caractéristiques** pour tous les caractères

Terminologie



Source: Wikipedia

1) hauteur x
2) ligne d'ascension
3) apex
4) ligne de base
5) ascension
6) croisement
7) stem
8) empattement
9) patte
10) bowl
11) counter
12) collet
13) boucle
14) oreille
15) cravate
16) ligne horizontale
17) bras
18) ligne verticale
19) hauteur de majuscule
20) ligne de descente

Formats

Polices matricielles

- Représentées à l'aide d'**images matricielles**
- La **définition** (résolution) est **fixe**
- Le rendu est **très rapide**
- Souvent utilisé dans les **jeux vidéos**

Polices vectorielles

- Plus **flexibles**, le rendu s'adapte au contexte
- Basées principalement sur les **courbes de Bézier**
- Polices **PostScript**: Adobe (Type 1 et Type 3), Metafont/Metapost
- Polices **TrueType** (extension `.ttf`): Apple et Microsoft
- Polices **OpenType** (extension `.otf`): plus récentes, effort conjoint

Classement des polices

Typographie

- Thibaudeau
- Vox-Atypi
- Novarese
- Alessandrini
- Chinoise, ...

W3C

- `serif`: à empattement
- `sans-serif`: sans empattement
- `cursive`: simule l'écriture manuscrite
- `fantasy`: décorative
- `monospace`: à largeur fixe

Créer des polices: FontForge

- Site officiel: http://fontforge.github.io/
- Licence: GPLv3
- La solution libre **principale** pour créer des fontes
- Excellent **tutoriel** disponible dans la documentation

Manipuler des images

Plusieurs traitements complémentaires

- **Pixel à pixel** (*point process*): traitement se fait sur chaque pixel, indépendamment de ses voisins
- **Convolutions et filtres** (*area process*): traitement d'un pixel dépend de ses voisins
- **Transformation géométriques** (*geometric process*): rotations, changements d'échelle, réflexions, interpolation, dilatation
- **Combinaison** d'images (*frame process*): addition, soustraction, mixte, masque, découpage
- **Traitement** de signal: domaine de fréquence (spectre), transformée de Fourier discrète, transformée du cosinus discrète
- **Compression** d'images...

Logiciels et applications

Applications graphiques

- Le traitement se fait en manipulant des **couches** (*layers*)
- Exemples: Gimp, Photoshop
- **Avantages**: rétroaction visuelle, plus intuitif, ne nécessite pas de connaissances en programmation

Par programmation

- En **ligne de commande**:

→ ImageMagick (`convert`, `montage`)

→ G'MIC = *GREYC's Magic for Image Computing*

- Ou avec des **bibliothèques graphiques**:

→ Cairo, Pillow, ImageMagick, libgmic, libpng

- **Avantages**: généralement plus rapide, permet de conserver une trace des opérations effectuées

Traitement pixel à pixel

- On agit sur chaque **pixel** indépendamment de ses voisins
- On peut aussi agir sur un seul **canal**:
- Généralement **rapide**: importation de l'image, traitement ponctuel, exportation de l'image
- Mathématiquement, il s'agit de **fonctions vectorielles** $\vec{T}$ de la forme

$$\begin{array}{rcl} \vec{T} : [0,1]^3 & \rightarrow & [0,1]^3 \\ (r,g,b) & \mapsto & (R(r,g,b), G(r,g,b), B(r,g,b)) \end{array}$$

- Par exemple, quelle est la **transformation** associée à la fonction suivante?

$$\vec{T}(r,g,b) = \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3} \right)$$

Exemples: ImageMagick

- **Inversion** de couleurs: $(r, g, b) \mapsto (1 - r, 1 - g, 1 - b)$

```
$ convert images/gimp-logo.png -negate negate.png
```

- Canal **rouge** à 1/2: $(r, g, b) \mapsto (1/2, g, b)$

```
$ convert images/gimp-logo.png -channel red \
    -fx "1/2" red.png
```

- **Colorisation** avec la couleur (r', g', b') avec proportions (x, y, z):

$$(r, g, b) \mapsto ((1 - x)r + xr', (1 - y)g + yg', (1 - z)b + zb')$$

```
$ convert images/gimp-logo.png -fill yellow \
    -colorize 50,50,50 gimp-logo-colorize.png
$ convert images/gimp-logo.png \
    -channel red   -fx "0.5 * u.r + 0.5 * 1.0" \
    -channel green -fx "0.5 * u.g + 0.5 * 1.0" \
    -channel blue  -fx "0.5 * u.b + 0.5 * 0.0" \
    gimp-logo-colorize2.png
```

Question sur les traitements pixel à pixel (1/2)

Exprimez chacune des transformations précédentes sous forme **matricielle**:

- $(r, g, b) \mapsto \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}\right)$

Question sur les traitements pixel à pixel (1/2)

Exprimez chacune des transformations précédentes sous forme **matricielle**:

- $(r, g, b) \mapsto \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}\right)$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

- $(r, g, b) \mapsto (1 - r, 1 - g, 1 - b)$

Question sur les traitements pixel à pixel (1/2)

Exprimez chacune des transformations précédentes sous forme **matricielle**:

- $(r, g, b) \mapsto \left(\frac{r+g+b}{3}, \frac{r+g+b}{3}, \frac{r+g+b}{3}\right)$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

- $(r, g, b) \mapsto (1 - r, 1 - g, 1 - b)$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

Question sur les traitements pixel à pixel (2/2)

- $(r, g, b) \mapsto (1/2, g, b)$

Question sur les traitements pixel à pixel (2/2)

- $(r, g, b) \mapsto (1/2, g, b)$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} 1/2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

- $(r, g, b) \mapsto ((1-x)r + xr', (1-y)g + yg', (1-z)b + zb')$

Question sur les traitements pixel à pixel (2/2)

- $(r, g, b) \mapsto (1/2, g, b)$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} 1/2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

- $(r, g, b) \mapsto ((1-x)r + xr', (1-y)g + yg', (1-z)b + zb')$

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} \mapsto \begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & z \end{bmatrix} \begin{bmatrix} r' \\ g' \\ b' \end{bmatrix} + \begin{bmatrix} 1-x & 0 & 0 \\ 0 & 1-y & 0 \\ 0 & 0 & 1-z \end{bmatrix} \begin{bmatrix} r \\ g \\ b \end{bmatrix}$$

Traitements géométriques

- Rotations (`-rotate <angle>`)
- Réflexions (`-flip`, `-flop`, `-transpose`)
- Changements d'échelle (`-scale`)

```
# Rotations
$ convert images/gimp-logo.png -rotate 20 rotate+20.png
$ convert images/gimp-logo.png -rotate -30 rotate-30.png

# Réflexions
$ convert images/gimp-logo.png -flip horizontal.png
$ convert images/gimp-logo.png -flop vertical.png
$ convert images/gimp-logo.png -transpose transpose.png

# Changements d'échelle
$ convert images/gimp-logo.png -scale '20%' minify.png
$ convert images/gimp-logo.png -scale '500%' magnify.png
```

Convolutions

- Chaque pixel est calculé en fonction de ses **voisins**
- Permet de créer différents **effets**

$\rightarrow$ flous

$\rightarrow$ détection d'arêtes

- L'idée est d'utiliser une **matrice carrée de poids**

$$W = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1} & w_{n,2} & \cdots & w_{n,n} \end{bmatrix}$$

- La matrice est généralement **d'ordre impair**
- Aussi, si la **somme** des poids est égale à 1, on préserve l'**intensité**
- La matrice est appelée **masque de convolution**

Balayage et limites

- Pour appliquer une **convolution**, on balaye chaque pixel
- Le nouveau pixel est obtenu en fonction de ses **voisins**
- En tenant compte du **masque**

Traitement des bordures

- Valeur par défaut aux pixels **hors image**
- Ne **pas toucher** aux pixels limites
- Supposer que l'image est **périodique**

Exemple: masque de flou

- Masques 3×3:

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}, \quad \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}$$

- Masques 5×5:

$$\begin{bmatrix} 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \\ 1/25 & 1/25 & 1/25 & 1/25 & 1/25 \end{bmatrix}$$

Exemple: masque de contraste (*sharpening*)

- Masques **communs**:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}, \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

- Masques **haute définition**:

$$\begin{bmatrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{bmatrix}$$

Exemples: convolutions avec ImageMagick

```
# Application d'un flou
$ convert images/gimp-logo.png -define convolve:scale=!\
    -morphology Convolve Octagon:2 gimp-logo-convolve.png

# Détection des arêtes
$ convert images/gimp-logo.png\
    -define convolve:scale='50%!' -bias '50%'\
    -morphology Convolve Sobel -solarize '50%'\
    -level '50,0%' gimp-logo-edges.png
```

On peut ensuite **combiner** les résultats:

```
$ composite -compose difference images/gimp-logo.png\
    gimp-logo-edges.png gimp-logo-difference.png
```