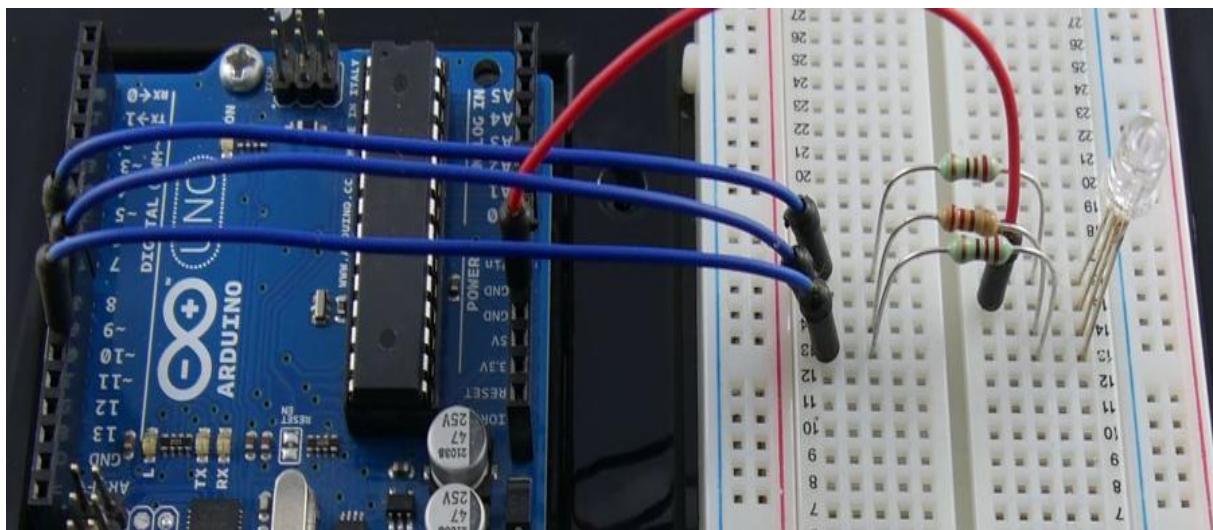


La diode L.E.D. avec Arduino



Cette diode émet de la lumière quand elle polarisée en direct. Elle est maintenant très utilisée pour l'éclairage des habitations et depuis 1975 comme voyants lumineux dans les appareils électroniques.

Principe de fonctionnement

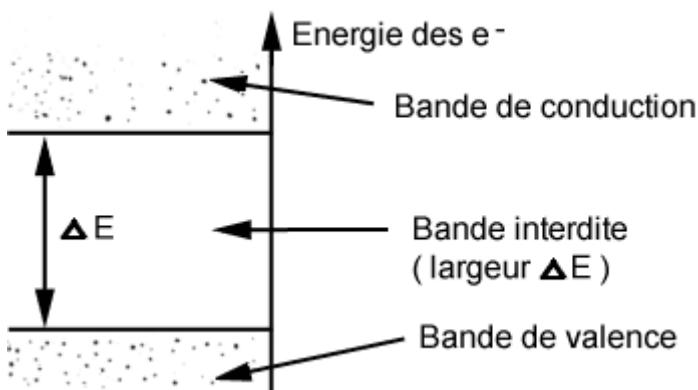
Le mot LED est l'acronyme de Light Emitting Diode (Diode Electroluminescente en français). Le symbole de la LED ressemble à celui de la diode mais on y a ajouté deux flèches sortantes pour représenter le rayonnement lumineux émis.



Symbole de la LED.

Electroluminescence

La physique des semi-conducteurs nous enseigne que les électrons dans les solides cristallins se situent à des niveaux d'énergie spécifiques. Ces niveaux très proches les uns des autres, sont regroupés en "bandes d'énergie".



Un électron de la bande de valence peut passer dans la bande de conduction à condition d'acquérir une énergie supplémentaire au moins égale à Delta E.
C'est l'effet photoélectrique.

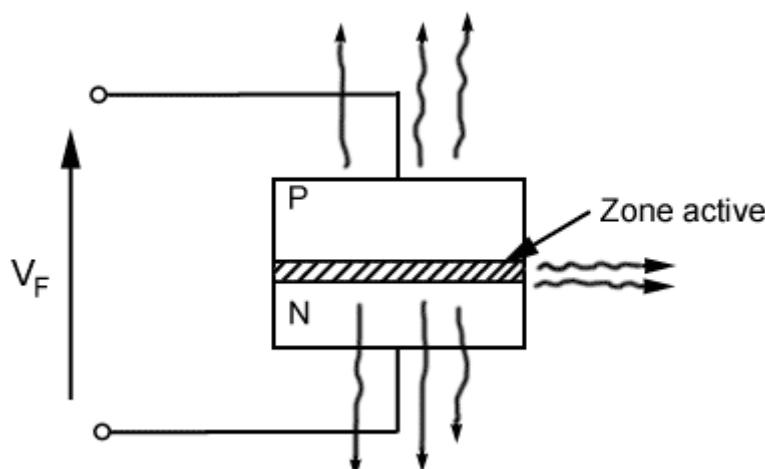
Un électron de la bande de conduction peut passer dans une bande de valence.
Dans ce cas il libère une énergie au moins égale à Delta E.
Cette énergie peut être :

- Dissipée sous forme de chaleur (phonons),
- émise sous forme de lumière (photons).

C'est l'effet électroluminescence (visible ou non).

Jonction P.N.

Ce phénomène d'électroluminescence sera obtenu à la condition de créer une forte quantité d'électrons dans la bande de conduction. On l'obtient par injection de porteurs en polarisant dans le sens direct, une jonction PN à semi-conducteur. Le même résultat aurait pu être obtenu en irradiant le cristal avec une source lumineuse d'énergie importante (photoluminescence) ou par bombardement électronique (cathodoluminescence).



Selon la fabrication, la lumière peut être émise soit latéralement, soit perpendiculairement à travers la mince couche N ou P.

Caractéristiques optiques

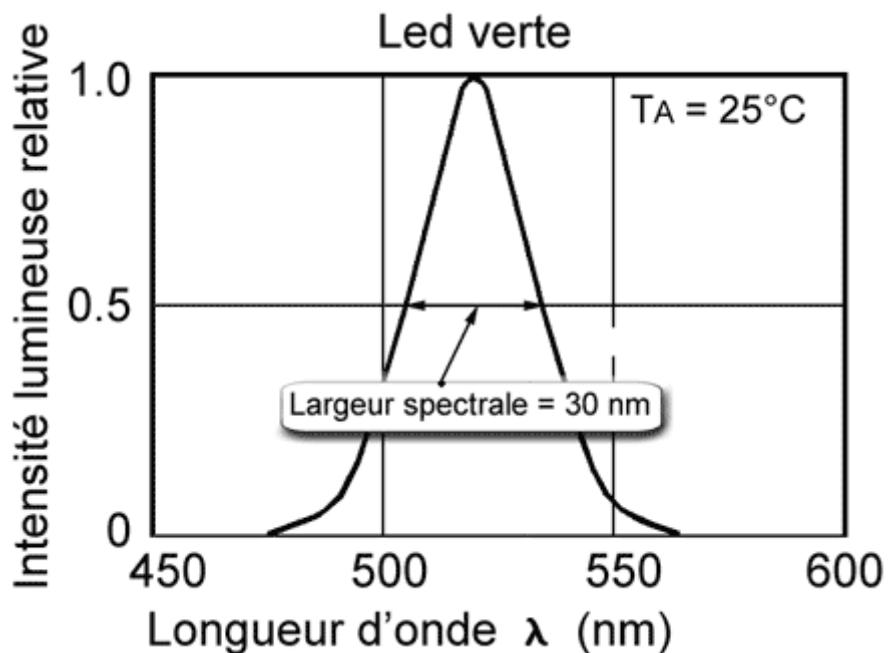
Longueur d'onde du pic d'émission

Cette valeur nous indique la longueur d'onde (λ_p), en nano-mètre, à laquelle est émis la plus importante partie du rayonnement (wavelength). La valeur est donnée pour une intensité de courant (I_F).

Spectre ou largeur spectrale à mi-intensité

Le spectre d'émission d'une diode LED est relativement étroit.

Exemple : pour une longueur d'onde à intensité maximale égale à 520 nm, la longueur d'onde à intensité moitié pourra être comprise de 505 nm à 535 nm (soit une largeur spectrale de 30 nanomètres).



Il existe actuellement plusieurs types de LED donnant chacun des spectres différents. Cela est obtenu par la variété des semi-conducteurs utilisés pour fabriquer les jonctions PN.

Exemples dans le tableau suivant pour l'obtention de certaines longueurs d'onde :

Matériaux	Rayonnement	
InAs	ultra-violet	315 nm
InP	infra-rouge	910 nm
GaAsP ₄	rouge	660 nm
GaAsP ₈₂	jaune	590 nm
GaP	vert	560 nm

Correspondance couleurs, longueurs d'onde et énergie des photons

Couleur	Longueur d'onde (nm)	
UltraViolet	< 390	> 3,18
Violet	390-455	2,72-3,18
Bleu	455-490	2,53-2,72
Cyan	490-515	2,41-2,53
Vert	515-570	2,18-2,41
Jaune	570-600	2,06-2,18
Orange	600-625	1,98-2,06
Rouge	625-720	1,72-1,98
InfraRouge	> 720	< 1,72

Diagramme de rayonnement

Le flux lumineux n'est pas homogène tout autour de la LED. La répartition spatiale de la puissance émise dépend de la forme de la diode LED :

- forme de la partie émissive (point, trait...),
- avec lentille de concentration ou sans,
- diffusante ou non.

Cette répartition est définie par le diagramme de rayonnement qui représente la répartition angulaire de l'intensité relative émise.

Exemple :

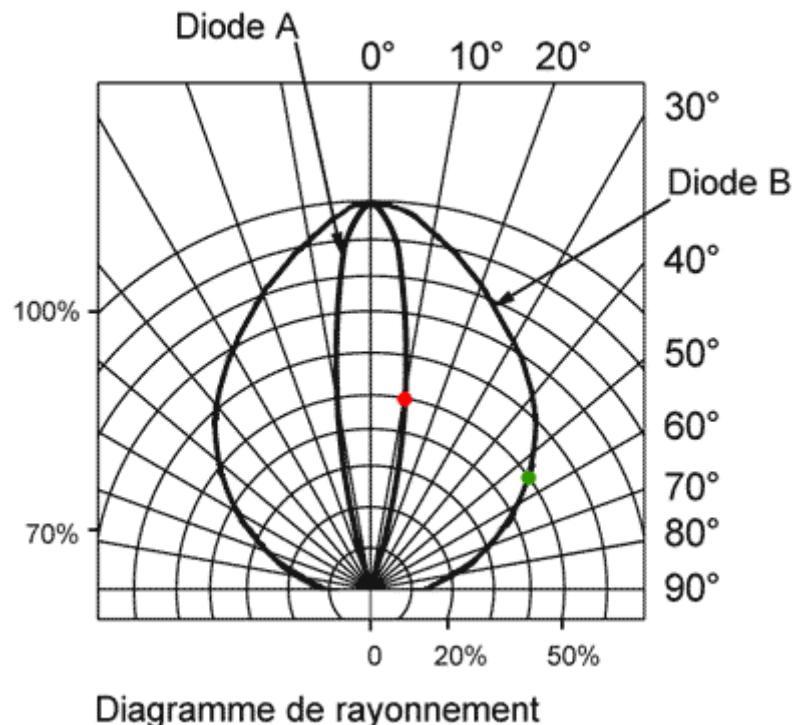


Diagramme de rayonnement

Angle d'émission à mi-intensité

Les fabricants précisent souvent l'angle pour lequel l'intensité lumineuse a été réduite de moitié.

Sur le diagramme ci-dessus, le point rouge indique un angle de 10 degrés et le point vert un angle de 50° pour une intensité relative émise de 50%.

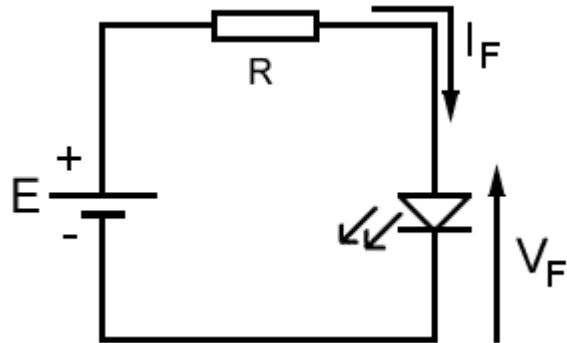
Intensité lumineuse

L'intensité lumineuse (mesurée en candelas) est la quantité de lumière émise dans une certaine direction à 1 mètre de distance. Dans les caractéristiques optiques des leds nous l'exprimons aussi en micro-candela (mcd) et se note λ .

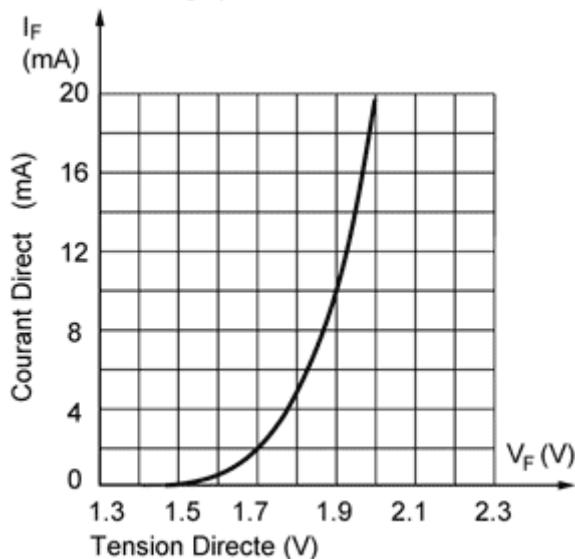
Caractéristiques électriques

Point de fonctionnement et tension direct

Une LED se comporte électriquement comme une diode. Pour émettre elle doit être polarisée en direct.



La caractéristique $I_F(V_F)$ montre que la tension de conduction de la diode LED (forward voltage) est environ 1,5 volts à 2 V.



Le courant I_F vaut environ $E-2V/R$.

En pratique, le constructeur préconise 10 à 20 mA.

Le courant traversant la LED détermine l'intensité lumineuse émise.

Remarque : certaines diodes ont des tensions de construction de l'ordre de 3 volts et plus.

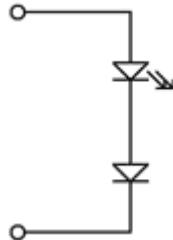
Tension inverse (V_R)

Dans certains cas, on peut avoir besoin de polariser en inverse la LED.

La diode est alors éteinte : elle n'émet plus d'intensité lumineuse.

Mais attention, la diode LED ne peut pas supporter des tensions inverses trop importantes comme une diode de redressement par exemple. Les valeurs courantes se situent telles que V_R max = ± 3 V à 5 V (reverse voltage) ; au delà de ces valeurs il y a endommagement ou destruction du composant. En cas de besoin nous plaçons

une diode normale en série avec la LED.



Courant direct en continu (I_F)

Le courant direct (mA) est donné en règle générale pour une température ambiante (TA) de 25°C. C'est le courant permanent que peut supporter la diode. Comme un semi-conducteur chauffe (avec agravement si TA > 25°C), il est recommandé de réduire l'intensité du courant (forward current).

Courant direct de crête (I_{FM})

C'est l'intensité d'une impulsion de courant direct maximum qui peut être appliquée à la LED pendant une durée déterminée. Entre deux impulsions de cette intensité, le composant doit avoir le temps de refroidir. Il faudra donc choisir un rapport entre durée d'impulsion et durée de pause assez grand.

Puissance et température de fonctionnement

La température de jonction doit rester inférieure à 125°C. Mais souvent les diodes LED sont montées dans des boîtiers plastiques. Dans ce cas, la température de fonctionnement ne doit pas dépasser 100°C. La puissance que peut dissiper une diode LED commune (ou utilisée en tant que témoin lumineux) est de l'ordre de 20 à 100 mw.

Les puissances des diodes LEDs destinées aux applications d'éclairage de locaux ou des lieux publics sont de l'ordre du Watt voir beaucoup plus quand il s'agit de module LED.

Influence de la tension directe

Toutes les LEDs présentent des variations de tension directe en fonction des changements de température de jonction. Le coefficient de température dépend du type de jonction. Les LEDs InGaAlP (jaune, orange et rouge) ont un coefficient compris entre -3,0 mV/K à -5,2 mV/K, et la LED InGaN (bleu, vert et blanc) ont un coefficient compris entre -3,6 mV/K et -5,2 mV/K.

Influence du courant I_F sur l'intensité lumineuse

L'œil est sensible à l'intensité lumineuse moyenne émise. L'intensité lumineuse donnée par le fabricant est obtenue dans des conditions de fonctionnement qu'il doit spécifier. Généralement il utilise un courant continu (à TA = 25° C).

D'autres valeurs de courant se traduisent par d'autres intensités lumineuses. En exploitant d'autres caractéristiques I_v (I_F) on s'aperçoit alors que l'intensité lumineuse augmente plus vite que le courant, c'est-à-dire que le rendement augmente pour un courant I_F , élevé mais bref, appelé courant de crête.

Voir l'exemple de la modulation d'une led infra-rouge pour télécommande RC5.

Il est alors extrêmement intéressant d'alimenter la LED en courant pulsé au lieu du courant continu. La valeur crête du courant permet alors d'obtenir des intensités lumineuses importantes. De ce fait nous pouvons :

- augmenter l'intensité lumineuse émise à consommation électrique moyenne égale,
- diminuer la consommation électrique tout en obtenant une intensité lumineuse égale,
- réduire l'échauffement de la jonction.

Influence de l'intensité lumineuse sur la température

L'intensité lumineuse diminue à mesure que la température augmente. Il s'agit d'un résultat de l'évolution des gains d'efficacité dans le semi-conducteur, et non le résultat de la variation de la tension direct en fonction de la température. Ce changement de température est non linéaire.

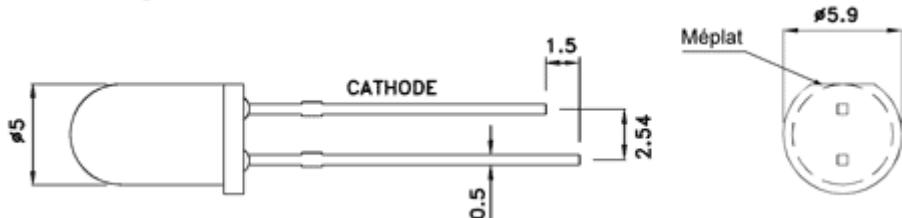
Décalage des coordonnées de chromaticité

Les caractéristiques de couleur des LED sont dépendantes du courant direct. Une attention particulière doit être accordée lors de l'utilisation des pilotes ou driver utilisés avec des LEDs RVB. Les gradateurs d'éclairage ne devraient pas modifier le rendu des couleurs. La solution préférée, est une gradation par PWM pour que chaque LED, de la composante RVB, soit pilotée avec le courant direct adapté.

Caractéristiques physiques

Composants traversants

Les fabricants proposent maintenant des leds de formes variées ; la plus commune de toutes étant la ronde. Elle se décline en plusieurs diamètre : 1,35 mm, 3 mm, 5 mm à 10mm. Nous trouvons également des led rectangulaires, triangulaires, carrées et en barre. Il faut bien repérer les connexions anode et cathode et respecter les consignes de mise en oeuvre lors de l'implantation de la led sur le circuit imprimé ou du soudage avec des fils.



Composants CMS

Sous cette forme les boîtiers sont moins encombrants et nous pouvons en souder plus sur une surface donnée. Ils conviennent pour la réalisation d'affichage, de feux de signalisation, modules électroniques miniatures ou une matrice de leds.



Réseau de LEDs

Les diodes électroluminescences discrètes peuvent être organisées en réseaux linéaires ou plan.

Dans le premier cas, elles peuvent remplacer un affichage analogique classique (galvanomètre).

Dans le second cas, elles serviront à toutes sortes d'affichages, y compris graphiques et leur commande sera généralement multiplexée.

Commande d'un réseau linéaire avec signal analogique

Après l'affichage par galvanomètre, puis l'affichage numérique, on trouve maintenant de plus en plus, un affichage mixte, où les valeurs analogiques sont quantifiées et affichées en échelle par tout ou rien.

C'est l'affichage analogique linéaire "bar-graph".

Formes

Les formes peuvent être variées :

- en ligne horizontale,
- verticale,
- multiple,
- circulaire simple,
- multiple.

Exemples de réseau linéaire à LEDs :

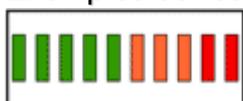
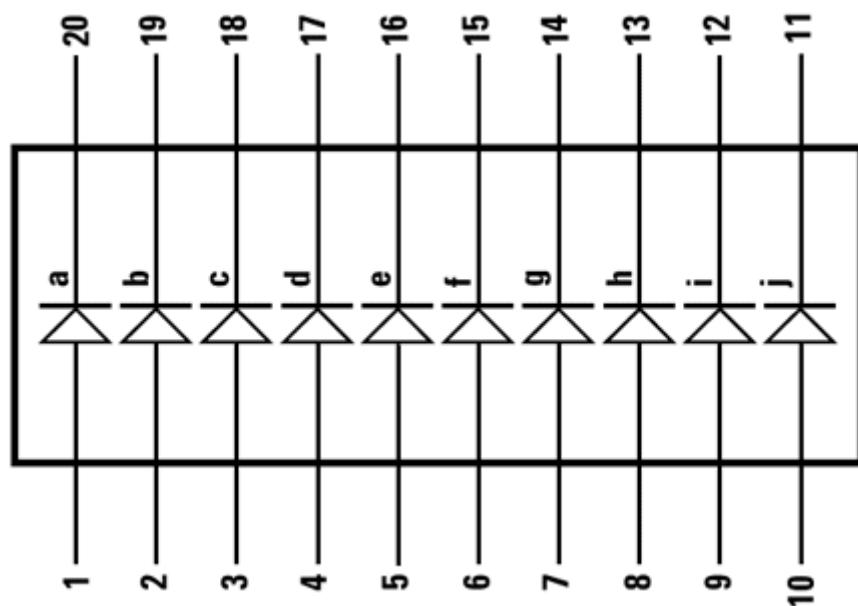
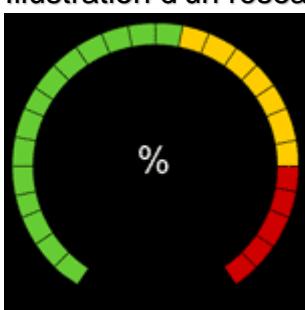


Schéma interne d'un réseau de leds :



sur cette configuration, on remarque bien la disponibilité de chacune des connexions des leds.

Illustration d'un réseau circulaire à LEDs :



L'affichage peut respecter toutes les lois mathématiques (logarithmiques par exemple) et être commandé à partir d'informations codées de toutes sortes.

Les LEDs sont de merveilleux petits composants électroniques. Il suffit de leur envoyer quelques volts de tension pour qu'elles illuminent (littéralement) votre journée.

On trouve des LEDs classiques monocouleur absolument partout. Mais il existe aussi des LEDs bicolores (on en parlera dans un autre article) et des LEDs à trois couleurs dites "RGB" (ou "RVB" en français, pour "Rouge Vert Bleu").

Ces LEDs RGB permettent de mettre de l'ambiance dans une pièce, ou à plus petite échelle, d'afficher une information en couleur. Dans cet article, nous allons apprendre à utiliser une LED RGB avec une carte Arduino / Genuino.

Les LEDs RGB



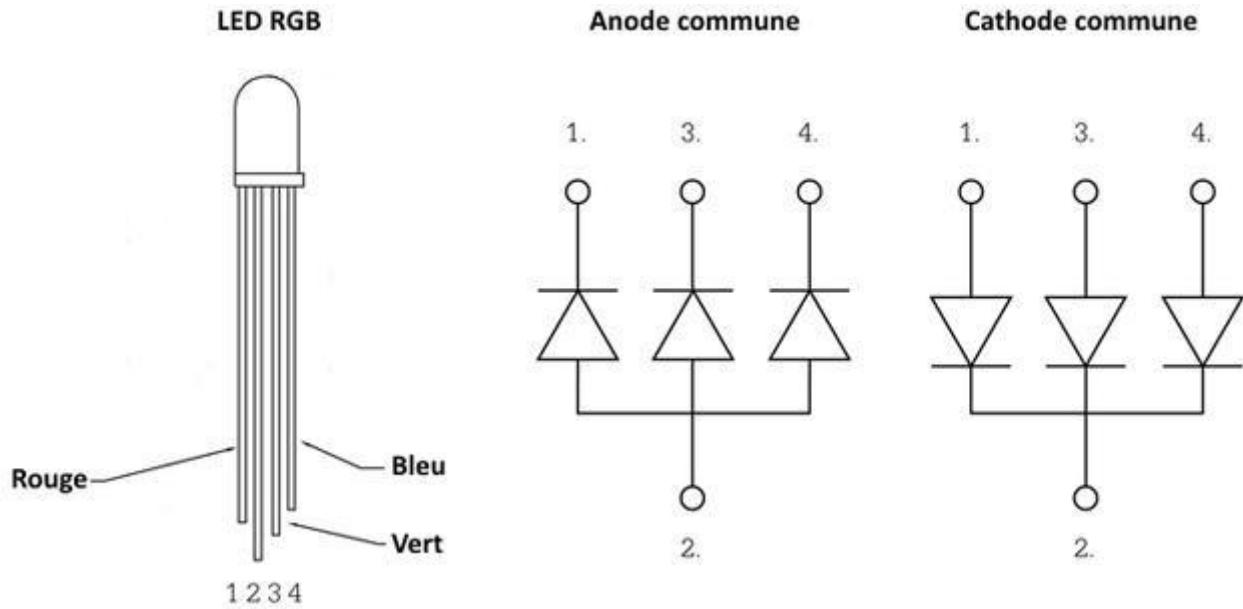
Photographie de plusieurs LEDs RGB

Les LEDs RGB sont en réalité composées de trois LEDs classiques Rouge, Verte et Bleu, emprisonnées ensemble pour l'éternité dans un même boîtier plastique.

Le boîtier des LEDs RGB existe en deux variantes : clair et diffusant. Le boîtier clair (voir photo ci-dessus) permet de voir les trois couleurs séparément et donne un effet assez sympathique. Le boîtier diffusant, au contraire, ne permet pas de voir les différentes couleurs, mais une seule couleur correspondant au mélange des trois couleurs de base. Il est donc important de bien choisir le type de boîtier en fonction de

son projet 😊

L'avantage d'une LED RGB par rapport à trois LEDs classiques est simple : il n'y a qu'un seul composant à câbler. Au lieu d'avoir trois composants à deux pattes, on a un unique composant à quatre pattes, ça demande moins de soudure et donc moins de temps à câbler.



Le brochage d'une LED RGB

Les LEDs RGB existent en deux versions : à anode commune ou à cathode commune.

Dans la version à anode commune, les trois anodes (le "+") des LEDs sont reliées ensemble. Cela signifie qu'il faut câbler la tension d'alimentation sur la broche commune et contrôler les LEDs via un signal à 0 volt pour les faire s'allumer.

Dans la version à cathode commune, les trois cathodes (le "-") des LEDs sont reliées ensemble. Cela signifie qu'il faut câbler la masse sur la broche commune et contrôler les LEDs via un signal à +5 volts (ou autre) pour les faire s'allumer.

Les versions à cathode commune sont plus simples à utiliser pour des débutants, car plus intuitives. Dans cette configuration, une tension de 5 volts allume la LED et une tension de 0 volt l'éteint. Pour un petit projet avec seulement quelques LEDs RGB, cela peut être intéressant.

Cependant, les versions à anode commune sont bien plus répandues. Elles sont moins simples à utiliser, car dans cette configuration une tension de 5 volts éteint la LED et une tension de 0 volt l'allume. C'est le monde à l'envers.

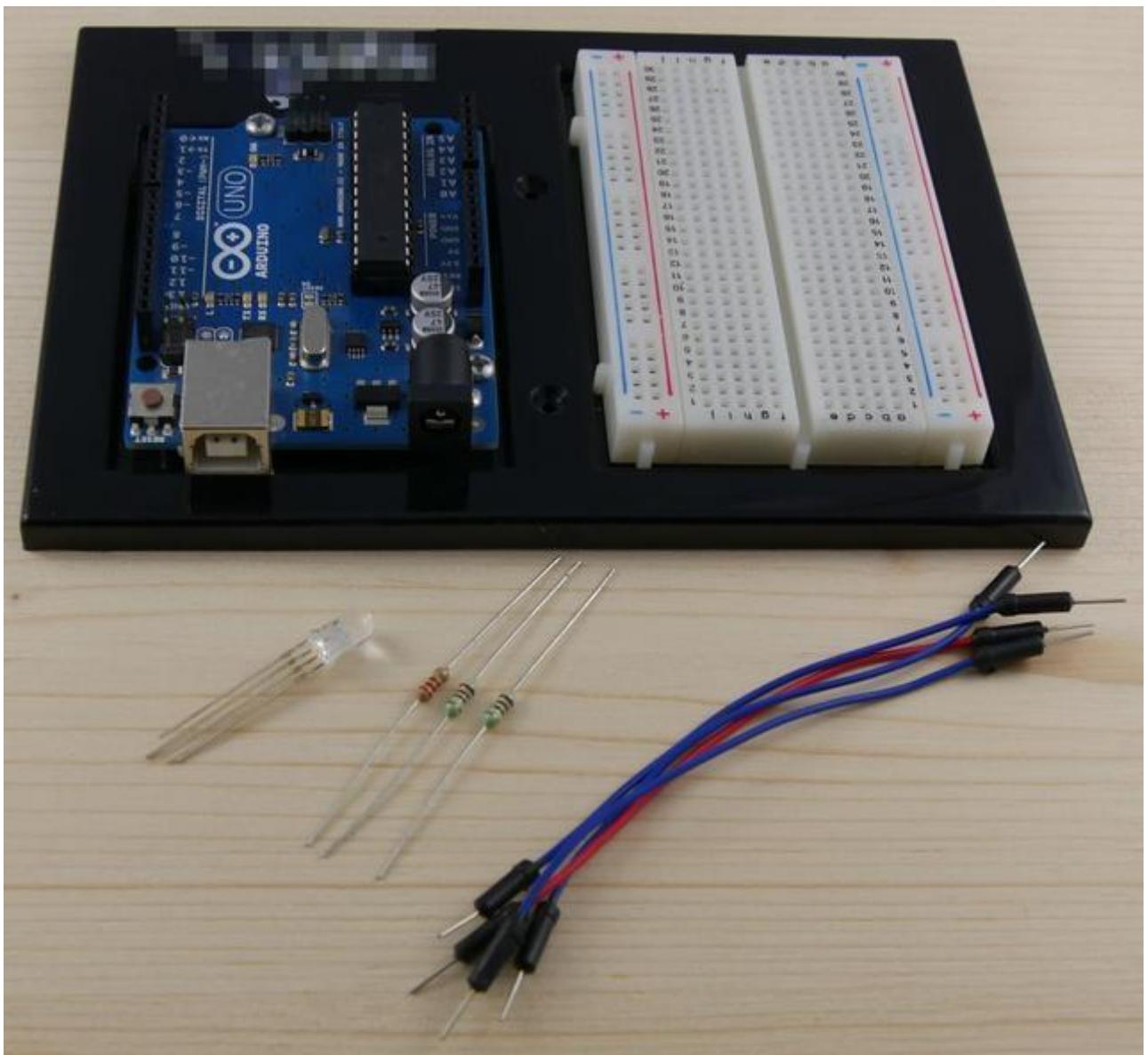
De façon générale, dans une vraie application, il est préférable d'utiliser des LEDs RGB à anode commune plutôt que des LEDs RGB à cathode commune. Elles sont certes moins pratiques à utiliser, car il faut inverser sa logique de pensées, mais ces versions ont l'immense avantage de pouvoir être contrôlées par des circuits intégrés spécialisés comme le TLC5940 qui ne peuvent qu'absorber du courant et pas en

générer, ce qui rend l'utilisation de LEDs RGB à cathode commune impossible dans ce cas.

Utiliser une LED RGB avec une carte Arduino / Genuino

Pour bien comprendre le fonctionnement d'une LED RGB, nous allons faire un montage très simple avec une LED RGB à anode commune et quelques résistances.

Le montage

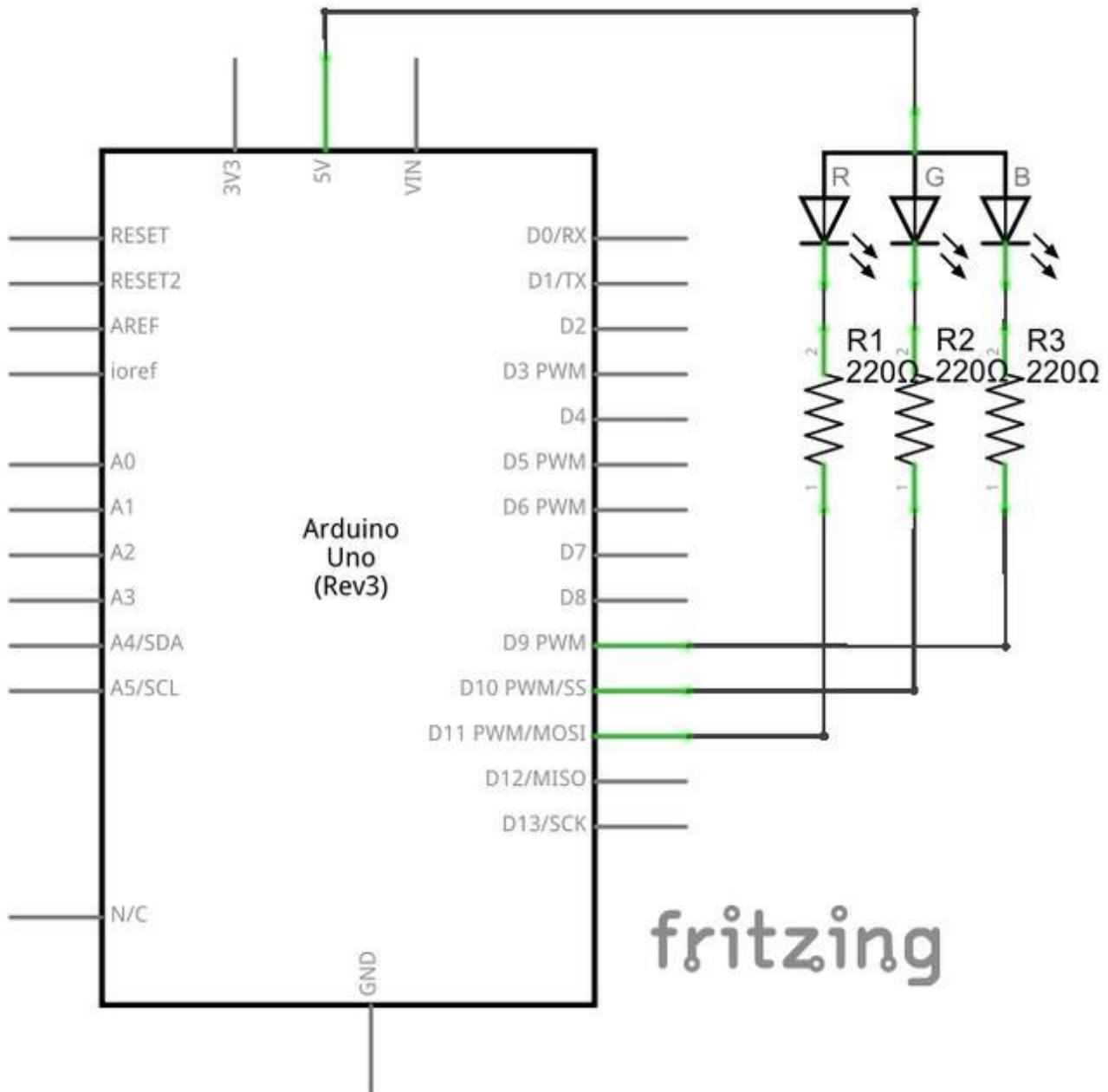


Matériel nécessaire

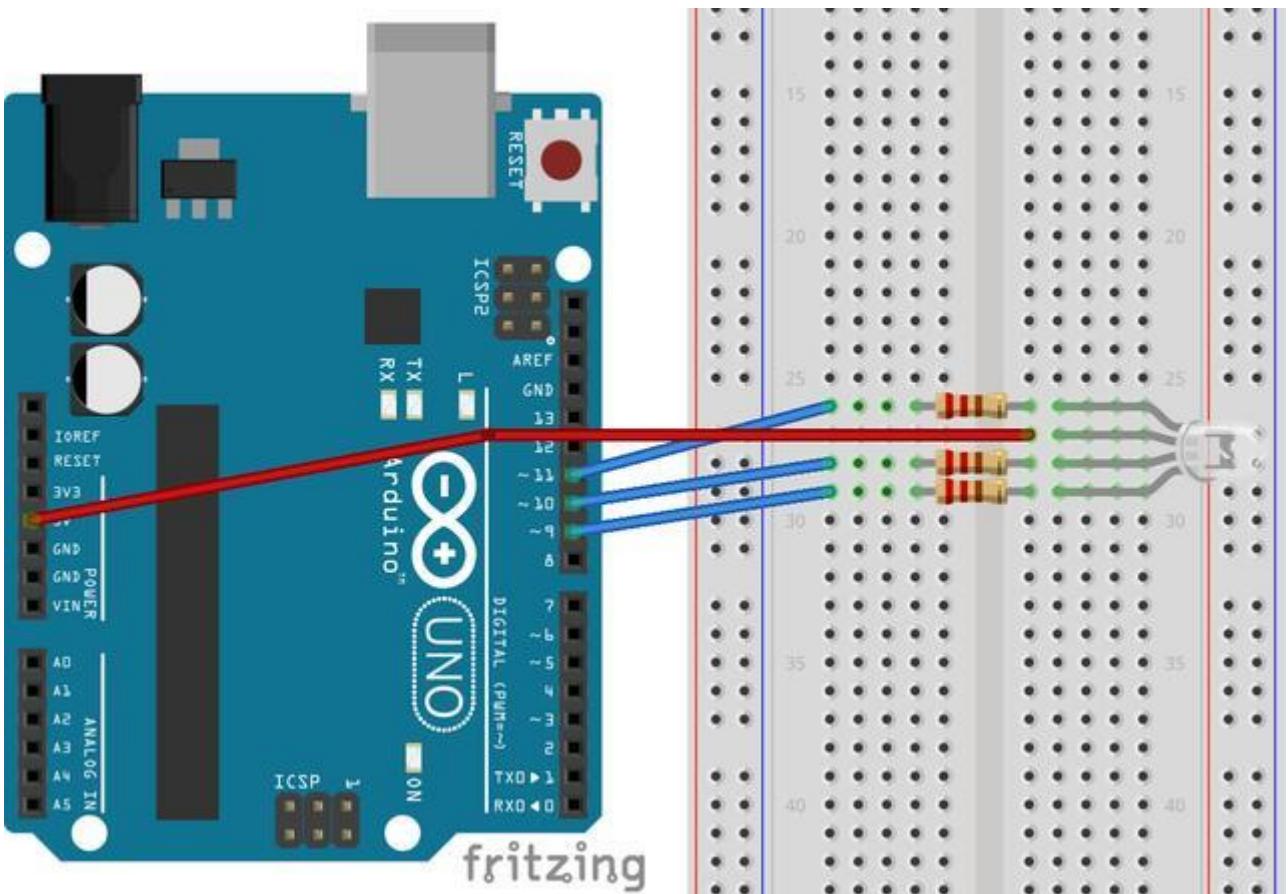
Pour réaliser ce montage, il va nous falloir :

- Une carte Arduino UNO (et son câble USB),

- Une LED RGB à anode commune (ou cathode commune, voir chapitre suivant dans ce cas),
- Trois résistances de 220 ohms (rouge / rouge / marron),
- Une plaque d'essai et des fils pour câbler notre montage.

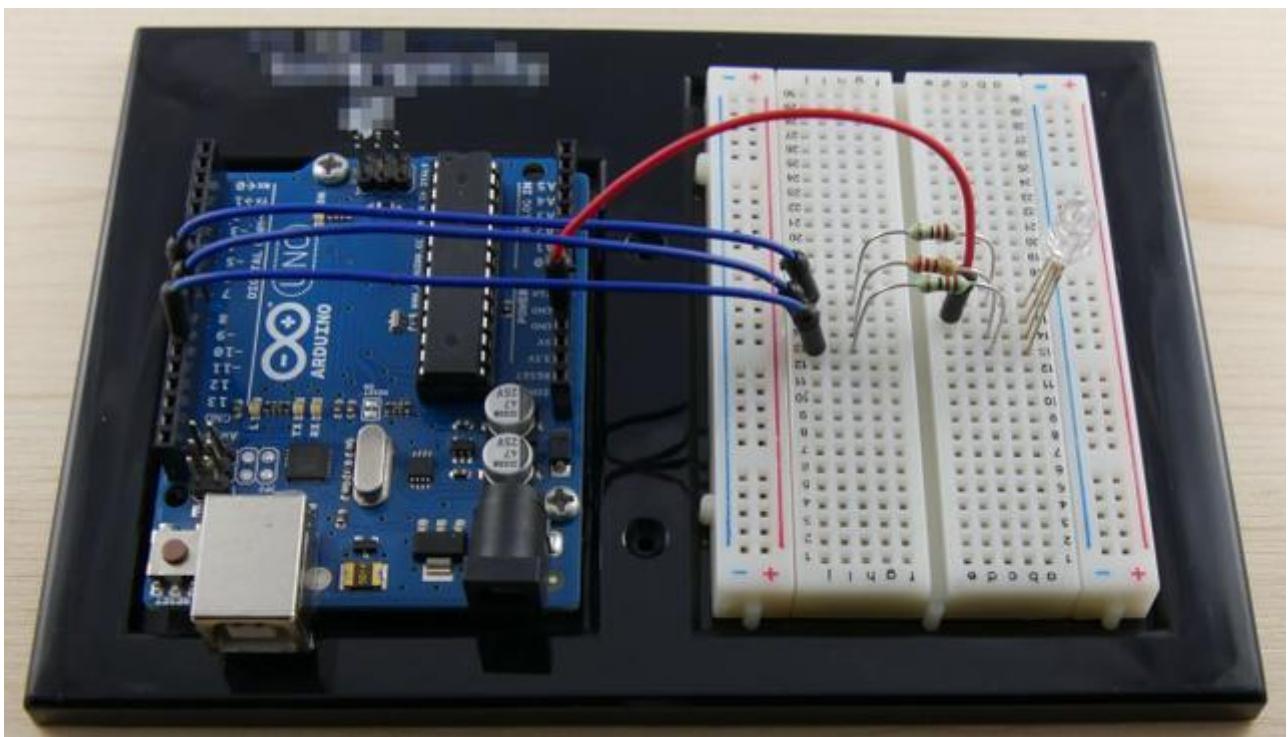


Vue schématique du montage



Vue prototypage du montage

Pour commencer notre montage, nous allons câbler la broche VCC de la carte Arduino à la broche commune de la LED RGB au moyen d'un fil. On relie ensuite ensuite chaque broche restante de la LED RGB à une résistance de 220 ohms.



Le montage fini

Pour finir, on câble chaque résistance respectivement aux broches D9, D10 et D11 de la carte Arduino.

N.B. Les broches utilisables avec la fonction `analogWrite()` (que l'on verra plus tard dans l'article) sont annotées avec un tilde (~) devant le nom de la broche.

Une LED, trois résistances

Pourquoi utiliser trois résistances ? Pourquoi ne pas utiliser une seule résistance sur la broche commune ? Ce serait plus simple !

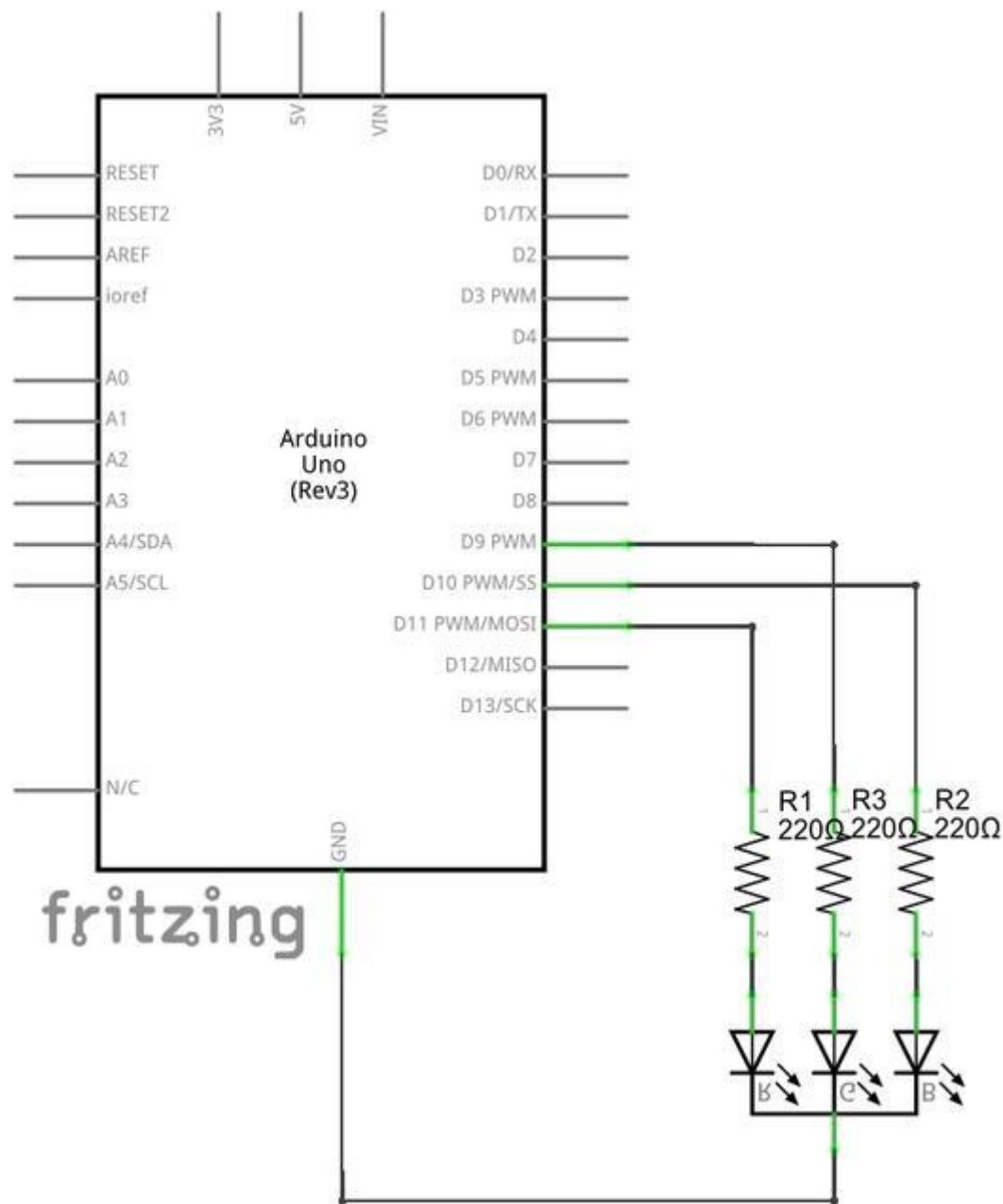
Oui, mais non, fausse bonne idée 😊 On câble toujours une résistance de limitation de courant par LED.

On pourrait effectivement utiliser une seule résistance sur la broche commune, mais cela aurait trois conséquences :

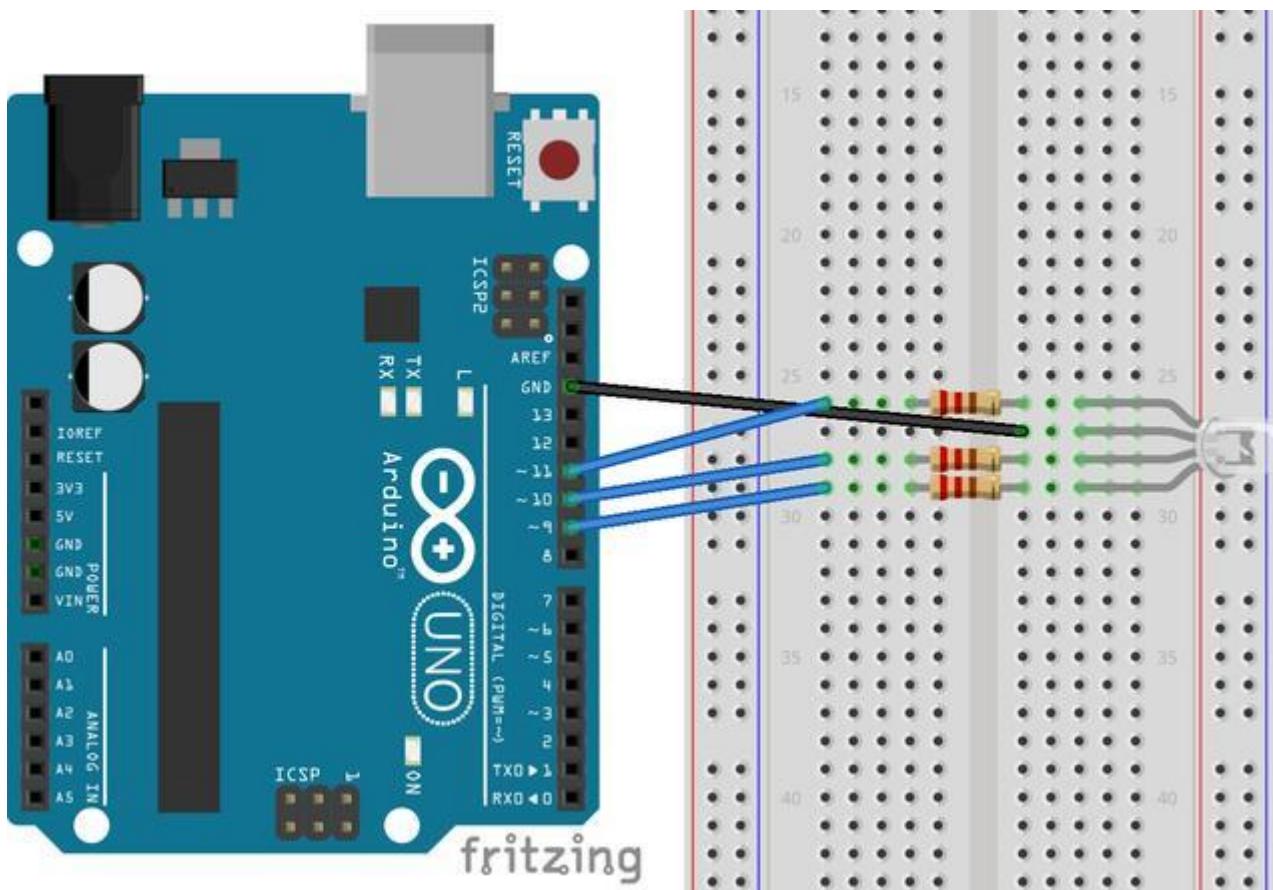
- allumer une seule LED lui ferait subir trop de courant,
- inversement, allumer plusieurs LED ferait diminuer la luminosité des LEDs,
- si une LED grille, toutes les autres suivront en cascade, car le courant restant deviendra plus fort à chaque LED défectueuse.

Une résistance ça ne coûte pas très cher, ne faites donc pas cette erreur de débutant 😊

Le montage (variante à cathode commune)



Vue schématique du montage (variante cathode commune)

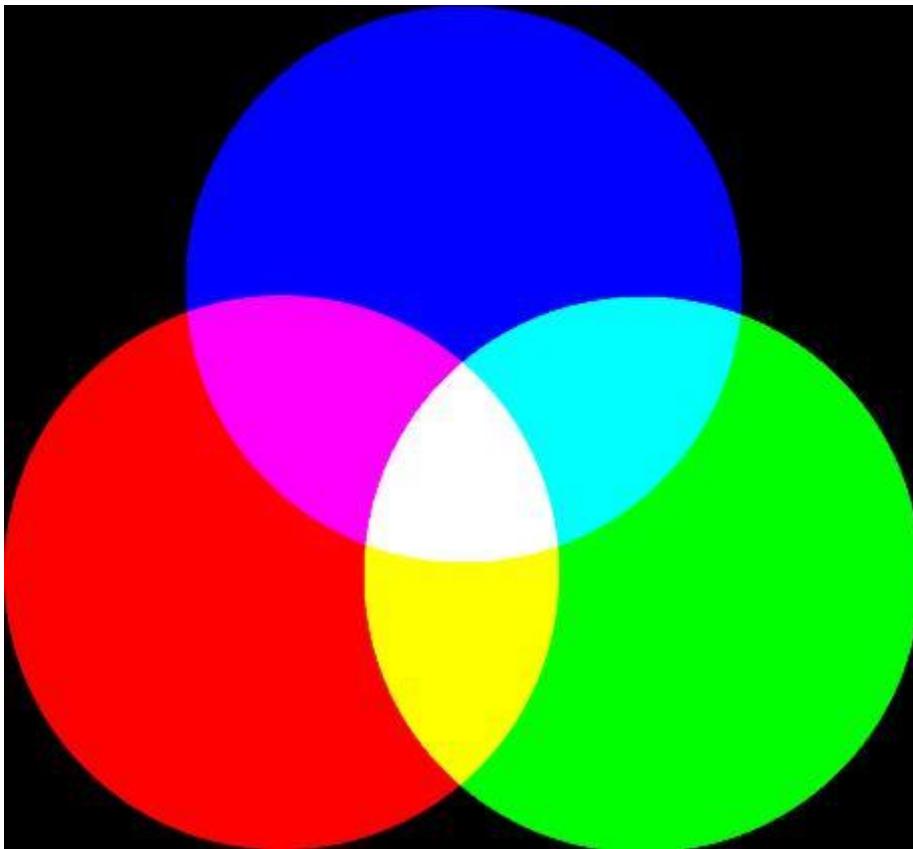


Vue prototypage du montage (variante cathode commune)

Le montage est identique à celui de la version à anode commune. La seule différence est que la broche commune de la LED RGB est reliée à la broche GND de la carte Arduino.

Le code V1

Commençons petit avec un code utilisant la fonction digitalWrite() pour contrôler chaque LED.



Couleurs primaires

Avec trois LED et deux états possibles par LED (éteinte et allumée), on obtient un total de 8 couleurs. Ça ne fait pas beaucoup de couleurs, mais c'est déjà un bon début.

```
1 /* Couleurs (format RGB) */
2 const byte COLOR_BLACK = 0b000;
3 const byte COLOR_RED = 0b100;
4 const byte COLOR_GREEN = 0b010;
5 const byte COLOR_BLUE = 0b001;
6 const byte COLOR_MAGENTA = 0b101;
7 const byte COLOR_CYAN = 0b011;
8 const byte COLOR_YELLOW = 0b110;
9 const byte COLOR_WHITE = 0b111;
10
11/* Broches */
12const byte PIN_LED_R = 9;
13const byte PIN_LED_G = 10;
```

```
14const byte PIN_LED_B = 11;
```

On commence le code très classiquement avec les déclarations des différentes constantes du programme.

Pour ce premier programme, j'ai décidé de coder les couleurs sur 3 bits, dans l'ordre R, G, B. Pour rendre cela plus lisible, j'ai utilisé le format 0bXXX qui est permis par le langage C/C++ pour déclarer des nombres en binaire.

J'ai ensuite déclaré trois constantes pour les trois broches de la LED RGB.

```
void setup() {
1
2  // Initialise les broches
3  pinMode(PIN_LED_R, OUTPUT);
4  pinMode(PIN_LED_G, OUTPUT);
5  pinMode(PIN_LED_B, OUTPUT);
6  displayColor(COLOR_BLACK);
7 }
8
```

Il n'y a pas grand-chose de très passionnant à faire dans la fonction `setup()`.

Il suffit de mettre les trois broches de la LED RGB en sortie et d'appeler notre fonction `displayColor()`, que l'on verra juste après, pour éteindre les LED au démarrage.

```
1 void displayColor(byte color) {
2
3  // Assigne l'état des broches
4  // Version cathode commune
5  //digitalWrite(PIN_LED_R, bitRead(color, 2));
6  //digitalWrite(PIN_LED_G, bitRead(color, 1));
7  //digitalWrite(PIN_LED_B, bitRead(color, 0));
8
9  // Version anode commune
```

```
10 digitalWrite(PIN_LED_R, !bitRead(color, 2));  
11 digitalWrite(PIN_LED_G, !bitRead(color, 1));  
12 digitalWrite(PIN_LED_B, !bitRead(color, 0));  
13}
```

La fonction `displayColor()` va faire tout le boulot. Dans cette première version à 8 couleurs, elle se contente de faire des `digitalWrite()` sur les broches des LEDs rouge verte et bleu en fonction de la couleur passée en paramètre.

Pour "lire" l'état de chaque bit de la couleur, j'utilise la fonction `bitRead()` qui prend en paramètre un nombre et un numéro de bit (commençant à 0).

N.B. Vous remarquerez qu'il y a deux versions du code, une pour les LEDs RGB à anode commune et une pour les LEDs RGB à cathode commune. La différence réside simplement dans l'inversion de valeur (le point d'exclamation signifie "inverse de la valeur booléenne", exemple : `!0 == 1`). A vous de commenter, décommenter la bonne

version du code en fonction de votre montage



```
1 void loop() {  
2  
3     /* Code de démonstration */  
4     displayColor(COLOR_RED);  
5     delay(1000);  
6  
7     displayColor(COLOR_GREEN);  
8     delay(1000);  
9  
10    displayColor(COLOR_BLUE);  
11    delay(1000);  
12  
13    displayColor(COLOR_MAGENTA);  
14    delay(1000);  
15  
16    displayColor(COLOR_CYAN);
```

```
17 delay(1000);
18
19 displayColor(COLOR_YELLOW);
20 delay(1000);
21
22 displayColor(COLOR_WHITE);
23 delay(1000);
24
25 displayColor(COLOR_BLACK);
26 delay(1000);
27}
```

La fonction `loop()` dans cet exemple se contente d'afficher chaque couleur en boucle avec un délai.

Le code complet avec commentaires :

```
1 /*
2  * Code d'exemple pour une LED RGB (8 couleurs).
3  */
4
5 /* Couleurs (format RGB) */
6 const byte COLOR_BLACK = 0b000;
7 const byte COLOR_RED = 0b100;
8 const byte COLOR_GREEN = 0b010;
9 const byte COLOR_BLUE = 0b001;
10 const byte COLOR_MAGENTA = 0b101;
11 const byte COLOR_CYAN = 0b011;
12 const byte COLOR_YELLOW = 0b110;
13 const byte COLOR_WHITE = 0b111;
14
```

```
15 /* Broches */
16 const byte PIN_LED_R = 9;
17 const byte PIN_LED_G = 10;
18 const byte PIN_LED_B = 11;
19
20 // Fonction setup(), appelée au démarrage de la carte Arduino
21 void setup() {
22
23 // Initialise les broches
24 pinMode(PIN_LED_R, OUTPUT);
25 pinMode(PIN_LED_G, OUTPUT);
26 pinMode(PIN_LED_B, OUTPUT);
27 displayColor(COLOR_BLACK);
28}
29
30 // Fonction loop(), appelée continuellement en boucle tant que la carte Arduino est alimentée
31
32 void loop() {
33
34     /* Code de démonstration */
35     displayColor(COLOR_RED);
36     delay(1000);
37
38     displayColor(COLOR_GREEN);
39     delay(1000);
40
41     displayColor(COLOR_BLUE);
42     delay(1000);
43
44     displayColor(COLOR_MAGENTA);
```

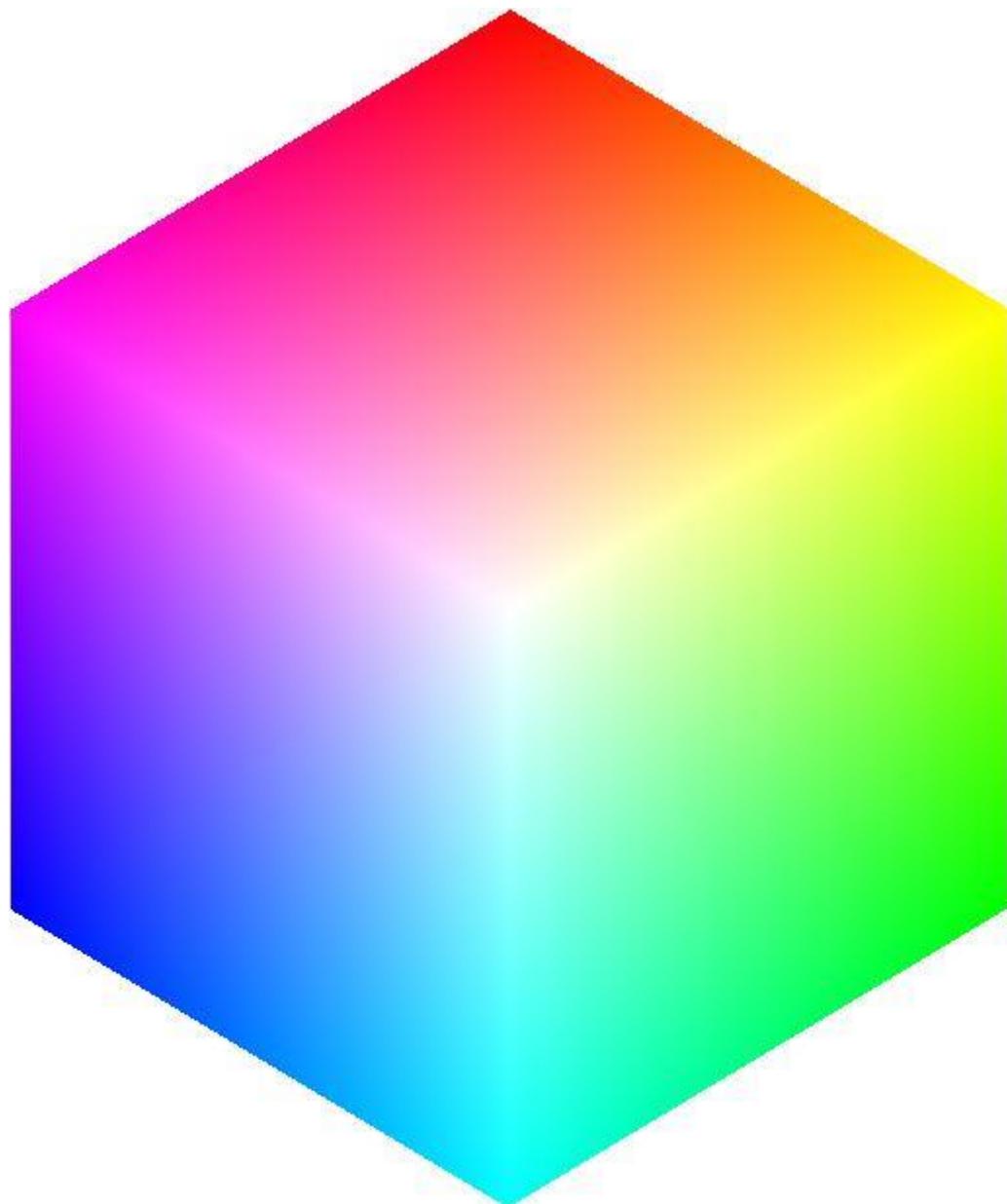
```
45 delay(1000);
46
47 displayColor(COLOR_CYAN);
48 delay(1000);
49
50 displayColor(COLOR_YELLOW);
51 delay(1000);
52
53 displayColor(COLOR_WHITE);
54 delay(1000);
55
56 displayColor(COLOR_BLACK);
57 delay(1000);
58}
59
60/** Affiche une couleur */
61 void displayColor(byte color) {
62
63 // Assigne l'état des broches
64 // Version cathode commune
65 //digitalWrite(PIN_LED_R, bitRead(color, 2));
66 //digitalWrite(PIN_LED_G, bitRead(color, 1));
67 //digitalWrite(PIN_LED_B, bitRead(color, 0));
68 // Version anode commune
69 digitalWrite(PIN_LED_R, !bitRead(color, 2));
70 digitalWrite(PIN_LED_G, !bitRead(color, 1));
71 digitalWrite(PIN_LED_B, !bitRead(color, 0));
}
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Le code V2

Huit couleurs c'est bien sympathique, mais c'est un peu léger. 16.7 millions de couleurs, ce serait déjà beaucoup mieux !

Pour atteindre ce nombre hallucinant de nuances de couleurs, nous allons faire un second code, utilisant la fonction [analogWrite\(\)](#) cette fois-ci.



Cube RGB "True color" (RGB88)

La fonction [analogWrite\(\)](#) fonctionne différemment de la fonction [digitalWrite\(\)](#), celle-ci permet d'avoir 254 niveaux intermédiaires entre LOW et HIGH (256 niveaux au total).

Cette fonction fonctionne grâce au principe de PWM, mais on en parlera dans un article dédié 😊

La fonction `analogWrite()` n'est disponible que sur les broches compatibles PWM, annotées avec un tilde (~) devant leurs noms sur la carte Arduino.

```
/* Broches */
1 const byte PIN_LED_R = 9;
2 const byte PIN_LED_G = 10;
3 const byte PIN_LED_B = 11;
4
```

Comme pour le code précédent, on commence par les déclarations de constantes. Dans cette version, il suffit de déclarer les trois broches de la LED RGB.

```
void setup() {
1
2 // Initialise les broches
3 pinMode(PIN_LED_R, OUTPUT);
4 pinMode(PIN_LED_G, OUTPUT);
5 pinMode(PIN_LED_B, OUTPUT);
6 displayColor(0, 0, 0);
7 }
8
```

La fonction `setup()` est quasiment identique à la version précédente, seul l'appel à la fonction `displayColor()` est différent.

```
1 void displayColor(byte r, byte g, byte b) {
2
3 // Assigne l'état des broches
4 // Version cathode commune
5 //analogWrite(PIN_LED_R, r);
6 //analogWrite(PIN_LED_G, g);
7 //analogWrite(PIN_LED_B, b);
```

```
8
9 // Version anode commune
10 analogWrite(PIN_LED_R, ~r);
11 analogWrite(PIN_LED_G, ~g);
12 analogWrite(PIN_LED_B, ~b);
13}
```

Premier gros changement : les couleurs sont données en paramètres séparément. Chaque composante de la couleur est représentée par un nombre sur 8 bits (un octet), cela fait au total 24 bits, soit 16 777 216 couleurs possibles.

Les appels à `digitalWrite()` sont remplacés par des appels à `analogWrite()`. La valeur de chaque composante de la couleur voulue est donnée directement en paramètre de `analogWrite()`.

Dans le cas des LEDs RGB à anode commune, il faut inverser la valeur. On pourrait faire `255 - r` par exemple, mais il existe une syntaxe plus courte pour cela : `~r`. L'opérateur tilde permet d'inverser l'état de chaque bit d'un nombre, par exemple : `~0b101 == 0b010`.

```
1 void loop() {
2
3 /* Code de démonstration */
4 displayColor(255, 0, 0);
5 delay(1000);
6
7 displayColor(0, 255, 0);
8 delay(1000);
9
10 displayColor(0, 0, 255);
11 delay(1000);
12
13 displayColor(255, 0, 255);
14 delay(1000);
```

```
15
16 displayColor(0, 255, 255);
17 delay(1000);
18
19 displayColor(255, 255, 0);
20 delay(1000);
21
22 displayColor(255, 255, 255);
23 delay(1000);
24
25 displayColor(0, 0, 0);
26 delay(1000);
27}
```

La fonction `loop()` se contente encore une fois d'afficher chaque couleur en boucle avec un délai.

PS Je vous laisse réfléchir comment mettre en place un effet de fondu ou de dégradé de couleur 

Le code complet avec commentaires :

```
1  /*
2   * Code d'exemple pour une LED RGB (+16 millions de couleurs).
3   */
4
5  /* Broches */
6  const byte PIN_LED_R = 9;
7  const byte PIN_LED_G = 10;
8  const byte PIN_LED_B = 11;
9
10 // Fonction setup(), appelée au démarrage de la carte Arduino
```

```
11 void setup() {  
12  
13 // Initialise les broches  
14 pinMode(PIN_LED_R, OUTPUT);  
15 pinMode(PIN_LED_G, OUTPUT);  
16 pinMode(PIN_LED_B, OUTPUT);  
17 displayColor(0, 0, 0);  
18 }  
19  
20 // Fonction loop(), appelée continuellement en boucle tant que la carte Arduino est alimentée  
21 void loop() {  
22  
23 /* Code de démonstration */  
24 displayColor(255, 0, 0);  
25 delay(1000);  
26  
27 displayColor(0, 255, 0);  
28 delay(1000);  
29  
30 displayColor(0, 0, 255);  
31 delay(1000);  
32  
33 displayColor(255, 0, 255);  
34 delay(1000);  
35  
36 displayColor(0, 255, 255);  
37 delay(1000);  
38  
39 displayColor(255, 255, 0);  
40
```

```

41   delay(1000);
42
43   displayColor(255, 255, 255);
44   delay(1000);
45
46   displayColor(0, 0, 0);
47   delay(1000);
48 }
49
50 /** Affiche une couleur */
51 void displayColor(byte r, byte g, byte b) {
52
53   // Assigne l'état des broches
54   // Version cathode commune
55   //analogWrite(PIN_LED_R, r);
56   //analogWrite(PIN_LED_G, g);
57   //analogWrite(PIN_LED_B, b);
58   // Version anode commune
59   digitalWrite(PIN_LED_R, ~r);
60   digitalWrite(PIN_LED_G, ~g);
61   digitalWrite(PIN_LED_B, ~b);
62 }
```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).

Bonus : Correction Gamma

Si vous jouez un peu avec le code du chapitre précédent, vous remarquerez sûrement que la luminosité des LEDs est bizarre.

Au début la luminosité semble s'accroire rapidement, puis d'un coup, la luminosité ne change plus. Cela est dû à la façon dont l'œil humain perçoit la lumière.

L'oeil humain est très fort quand il s'agit de détecter de toutes petites nuances de couleur ou de luminosité, mais seulement quand la luminosité est faible. Quand la luminosité dépasse un certain seuil, on ne remarque plus aucune nuance.

Cela est très embêtant, car les écrans d'ordinateur, les afficheurs, les LEDs, etc, sont capables d'afficher n'importe quelle nuance de couleur et/ou de luminosité de la même façon, qu'il importe la luminosité demandée. Pour un écran d'ordinateur, il est aussi simple d'afficher une transition de couleurs RGB(0, 0, 0) -> RGB(1, 1, 1) ou RGB(254, 254, 254) -> RGB(255, 255, 255) par exemple.



Image de test avec et sans correction Gamma

Il a donc fallu trouver une solution à ce problème. Cette solution s'appelle la "correction gamma". Le principe est simple : compenser la non-linéarité de la perception de la luminosité par l'oeil humain en application une correction à l'affichage.

L'image ci-dessus (qui au passage est une photo de test standard nommée "Lenna", du nom de l'actrice du magazine Playboy qui y est représentée – ce n'est pas une blague, on utilise une photo d'un magazine de fesse comme image de test standard depuis 1973) permet de bien voir l'effet de la correction gamma. Sans correction gamma, les couleurs paraissent délavées, fades.

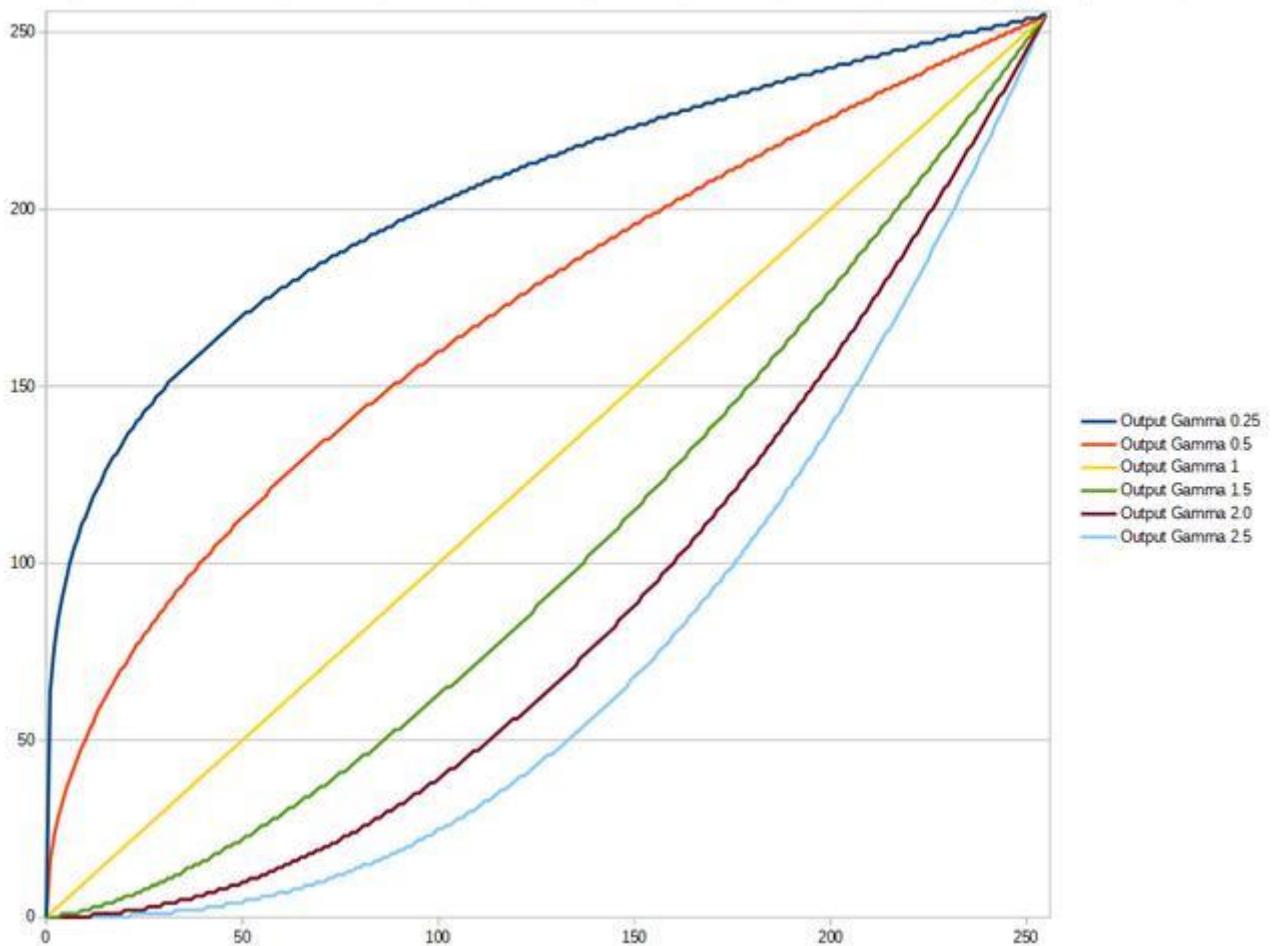


Illustration de différentes courbes de correction Gamma

Pour les amateurs de formules mathématiques, la fonction de courbe Gamma est : $\text{sortie} = \text{valeur_sortie_max} * \text{entrée} / \text{valeur_entrée_max}^{1/\text{coeff}}$. Le coefficient standard de correction est de 2.2, mais il existe d'autres standards comme 2.5. La courbe ci-dessus présente différentes valeurs de correction. Les valeurs supérieures à zéro corrigent le Gamma (plus sombre), les valeurs inférieures à zéro (dé)corrigent le Gamma (plus clair).

Afin de simplifier la correction Gamma en électronique, on utilise généralement une table de correction précalculée. Il suffit de donner une valeur en entrée de la table pour obtenir sa version corrigée en retour.

J'ai conçu un script [Python 3](#) qui permet de générer un fichier C/C++ prêt à l'usage en fonction du coefficient de correction désiré :

```

1  """
2  Look-up table Gamma correction generator for Python 3.
3  """
4

```

```
5  from math import pow
6
7
8  def gamma_lut(input_array_size, ouput_array_size=None,
9      correction_factor=2.2, reverse_gamma=False):
10     """
11     Generate a Gamma correction LookUp Table from the given parameters.
12     :param input_array_size: The input array size.
13     :param ouput_array_size: The output array size (default to the input array size).
14     :param correction_factor: The correction factor (between 0 and 3, included, default
15     2.2).
16     :param reverse_gamma: Set to `True` for reverse Gamma correction (default `Fals
17     e`).
18     :return: A generator expression.
19     """
20
21     assert input_array_size > 0, "Input array size must be positive"
22     assert ouput_array_size is None or ouput_array_size > 0, "Output array must be p
23 ositive"
24
25     # Fix output array size
26     ouput_array_size = ouput_array_size or input_array_size
27
28     # fix correction factor
29     if reverse_gamma:
30         correction_factor = 1.0 / correction_factor
31
32     # Gamma correction formula
33     for index in range(input_array_size):
34         yield round(pow(index / float(input_array_size - 1), correction_factor) * (ouput_a
rray_size - 1))
```

```
35
36
37 def split_chunks(array, chunk_size):
38     """ Split the given array into chunks of at most the given size. """
39     return [array[x:x + chunk_size] for x in range(0, len(array), chunk_size)]
40
41
42 # Output array size - MUST be a power of two
43 INPUT_ARRAY_SIZE = 256
44 OUTPUT_ARRAY_SIZE = 256
45
46 # Gamma correction coefficient - standard: 2.2, Maxim standard: 2.5
47 GAMMA_CORRECTION_COEFF = 2.2
48
49 # Number of values per line
50 NUMBER_COUNT_PER_LINE = 16
51
52 # Output header
53 print("""/* ----- BEGIN OF AUTO-GENERATED CODE - DO NOT EDIT ----- */
54 #ifndef GAMMA_H_
55 #define GAMMA_H_
56
57 /* Dependency for PROGMEM */
58 #include <avr/pgmspace.h>
59
60 /** Gamma correction table in flash memory. */
61 static const uint8_t PROGMEM _gamma[] = {"""
62
63 # Print LUT data
64
```

```

65 gamma_table = gamma_lut(INPUT_ARRAY_SIZE, OUTPUT_ARRAY_SIZE, GAMM
66 A_CORRECTION_COEFF)
67 gamma_table = [format(x, "3d") for x in gamma_table]
68 gamma_table_lines = split_chunks(gamma_table, NUMBER_COUNT_PER_LINE)
69 gamma_table_lines = [' ' + ', '.join(line) for line in gamma_table_lines]
70 print('\n'.join(gamma_table_lines))

71
72 # Output footer
73 print(""""

74 /**
75 * Apply gamma correction to the given sample.
76 *
77 * @param x The input 8 bits sample value.
78 * @return The output 8 bits sample value with gamma correction.
79 */
80
81 static inline uint8_t gamma(uint8_t x) {
82     return pgm_read_byte(&_gamma[x]);
83 }

#endif /* GAMMA_H_ */
/* ---- END OF AUTO-GENERATED CODE ---- */
"""

```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#).

Voici un exemple de table de correction 8 bits (entrée) vers 8 bits (sortie) avec une coefficient standard de 2.2 :

```

1 /* ---- BEGIN OF AUTO-GENERATED CODE - DO NOT EDIT ---- */
2 #ifndef GAMMA_H_
3 #define GAMMA_H_

```

4

5 /* Dependency for PROGMEM */

6 #include <avr/pgmspace.h>

7

8 /** Gamma correction table in flash memory. */

9 static const uint8_t PROGMEM _gamma[] = {

10 0, 21, 28, 34, 39, 43, 46, 50, 53, 56, 59, 61, 64, 66, 68, 70,
11 72, 74, 76, 78, 80, 82, 84, 85, 87, 89, 90, 92, 93, 95, 96, 98,
12 99, 101, 102, 103, 105, 106, 107, 109, 110, 111, 112, 114, 115, 116, 117, 118,
13 119, 120, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
14 136, 137, 138, 139, 140, 141, 142, 143, 144, 144, 145, 146, 147, 148, 149, 150,
15 151, 151, 152, 153, 154, 155, 156, 156, 157, 158, 159, 160, 160, 161, 162, 163,
16 164, 164, 165, 166, 167, 167, 168, 169, 170, 170, 171, 172, 173, 173, 174, 175,
17 175, 176, 177, 178, 178, 179, 180, 180, 181, 182, 182, 183, 184, 184, 185, 186,
18 186, 187, 188, 188, 189, 190, 190, 191, 192, 192, 193, 194, 194, 195, 195, 196,
19 197, 197, 198, 199, 199, 200, 200, 201, 202, 202, 203, 203, 204, 205, 205, 206,
20 206, 207, 207, 208, 209, 209, 210, 210, 211, 212, 212, 213, 213, 214, 214, 215,
21 215, 216, 217, 217, 218, 218, 219, 219, 220, 220, 221, 221, 222, 223, 223, 224,
22 224, 225, 225, 226, 226, 227, 227, 228, 228, 229, 229, 230, 230, 231, 231, 232,
23 232, 233, 233, 234, 234, 235, 235, 236, 236, 237, 237, 238, 238, 239, 239, 240,
24 240, 241, 241, 242, 242, 243, 243, 244, 244, 245, 245, 246, 246, 247, 247, 248,
25 248, 249, 249, 249, 250, 250, 251, 251, 252, 252, 253, 253, 254, 254, 255, 255
26};

27

28/**

29 * Apply gamma correction to the given sample.

30 *

31 * @param x The input 8 bits sample value.

32 * @return The output 8 bits sample value with gamma correction.

33 */

```

34static inline uint8_t gamma(uint8_t x) {
35    return pgm_read_byte(&_gamma[x]);
36}
37
38#endif /* GAMMA_H */
39/* ----- END OF AUTO-GENERATED CODE ----- */

```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#).

Pour démontrer l'utilité de la correction Gamma, voici un code de démonstration qui allume progressivement la LED rouge avec puis sans correction Gamma :

```

1  /*
2   * Code d'exemple pour une LED RGB (+16 millions couleurs) avec correction Gam
3   * ma.
4
5   /* Broches */
6   const byte PIN_LED_R = 9;
7   const byte PIN_LED_G = 10;
8   const byte PIN_LED_B = 11;
9
10  /* ----- BEGIN OF AUTO-GENERATED CODE - DO NOT EDIT ----- */
11 #ifndef GAMMA_H_
12 #define GAMMA_H_
13
14  /* Dependency for PROGMEM */
15 #include <avr/pgmspace.h>
16
17  /** Gamma correction table in flash memory. */
18 static const uint8_t PROGMEM _gamma[] = {
19     0, 21, 28, 34, 39, 43, 46, 50, 53, 56, 59, 61, 64, 66, 68, 70,

```

```

20 72, 74, 76, 78, 80, 82, 84, 85, 87, 89, 90, 92, 93, 95, 96, 98,
21 99, 101, 102, 103, 105, 106, 107, 109, 110, 111, 112, 114, 115, 116, 117, 118,
22 119, 120, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135,
23 136, 137, 138, 139, 140, 141, 142, 143, 144, 144, 145, 146, 147, 148, 149, 150,
24 151, 151, 152, 153, 154, 155, 156, 156, 157, 158, 159, 160, 160, 161, 162, 163,
25 164, 164, 165, 166, 167, 167, 168, 169, 170, 170, 171, 172, 173, 173, 174, 175,
26 175, 176, 177, 178, 178, 179, 180, 180, 181, 182, 182, 183, 184, 184, 185, 186,
27 186, 187, 188, 188, 189, 190, 190, 191, 192, 192, 193, 194, 194, 195, 195, 196,
28 197, 197, 198, 199, 199, 200, 200, 201, 202, 202, 203, 203, 204, 205, 205, 206,
29 206, 207, 207, 208, 209, 209, 210, 210, 211, 212, 212, 213, 213, 214, 214, 215,
30 215, 216, 217, 217, 218, 218, 219, 219, 220, 220, 221, 221, 222, 223, 223, 224,
31 224, 225, 225, 226, 226, 227, 227, 228, 228, 229, 229, 230, 230, 231, 231, 232,
32 232, 233, 233, 234, 234, 235, 235, 236, 236, 237, 237, 238, 238, 239, 239, 240,
33 240, 241, 241, 242, 242, 243, 243, 244, 244, 244, 245, 245, 246, 246, 246, 247, 247, 248,
34 248, 249, 249, 249, 250, 250, 251, 251, 252, 252, 253, 253, 254, 254, 255, 255
35 };
36
37 /**
38 * Apply gamma correction to the given sample.
39 *
40 * @param x The input 8 bits sample value.
41 * @return The output 8 bits sample value with gamma correction.
42 */
43 static inline uint8_t gamma(uint8_t x) {
44     return pgm_read_byte(&_gamma[x]);
45 }
46
47 #endif /* GAMMA_H_ */
48 /* ---- END OF AUTO-GENERATED CODE ---- */
49

```

```
50 // Fonction setup(), appelée au démarrage de la carte Arduino
51 void setup() {
52
53     // Initialise les broches
54     pinMode(PIN_LED_R, OUTPUT);
55     pinMode(PIN_LED_G, OUTPUT);
56     pinMode(PIN_LED_B, OUTPUT);
57     displayColor(0, 0, 0);
58 }
59
60 // Fonction loop(), appelée continuellement en boucle tant que la carte Arduino est allumée
61
62 void loop() {
63
64     /* Code de démonstration */
65     for (int i = 0; i < 256; i++) {
66         displayColor(i & 255, 0, 0);
67         delay(10);
68     }
69     delay(1000);
70
71     for (int i = 0; i < 256; i++) {
72         displayColor(gamma(i & 255), 0, 0);
73         delay(10);
74     }
75     delay(1000);
76 }
77
78     /** Affiche une couleur */
79 void displayColor(byte r, byte g, byte b) {
```

```
80
81 // Assigne l'état des broches
82 // Version cathode commune
83 //analogWrite(PIN_LED_R, r);
84 //analogWrite(PIN_LED_G, g);
85 //analogWrite(PIN_LED_B, b);
86 // Version anode commune
87 digitalWrite(PIN_LED_R, ~r);
88 digitalWrite(PIN_LED_G, ~g);
89 digitalWrite(PIN_LED_B, ~b);
}
```

L'extrait de code ci-dessus est disponible en téléchargement sur [cette page](#) (le lien de téléchargement en .zip contient le projet Arduino prêt à l'emploi).