Les bitmaps

Présentation par Renaud Pradenc

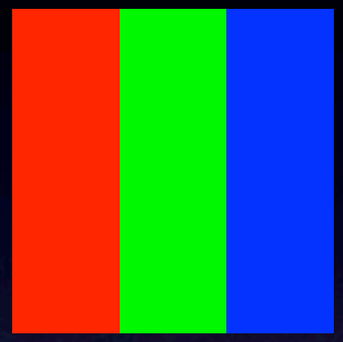
CÉROCE

Au programme

1. Généralités sur le graphisme bitmap
2. Organisation des bitmaps en mémoire
3. Création des bitmaps avec Core Graphics
4. Génération d'images
5. Effets sur les images

Généralités

Le pixel



- Un pixel est composé de trois sous-pixels de couleurs rouge, vert et bleu
- Varier l'intensité lumineuse de ces trois segments permet de régler la couleur

La bitmap

- Une bitmap est une grille de pixels
- Les bitmaps sont omni-présentes: tout ce qui est affiché à l'écran, toutes les images numérisées

Graphismes vectoriels

- Ils sont définis par des vecteurs plutôt que par des pixels
- Ils doivent être *rasterisés* pour être affichés

Pourquoi manipuler les bitmaps ?

- Pour appliquer des effets
- Pour générer des images
- Pour en extraire de l'information
- Pour implémenter des systèmes de cache

Organisation en mémoire

Organisation de base

Quelle est l'adresse d'un pixel d'après ses coordonnées ?

- la *même quantité* de mémoire est allouée à chaque pixel
- les pixels se suivent en mémoire
- la bitmap se lit de gauche à droite, et de haut en bas

Exemple: 8 bits/pixel

- 8 bits/pixel = 256 niveaux de gris
  (0 = noir, 255= blanc)
- Dimensions: 15 x 10 pixels

adrBitmap

15 colonnes

10 lignes

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
15	16	17	…											
30														
45														
60														
75														
90														
105														
120														
135														149

adrPixel = adrBitmap + ligne × largeur + colonne

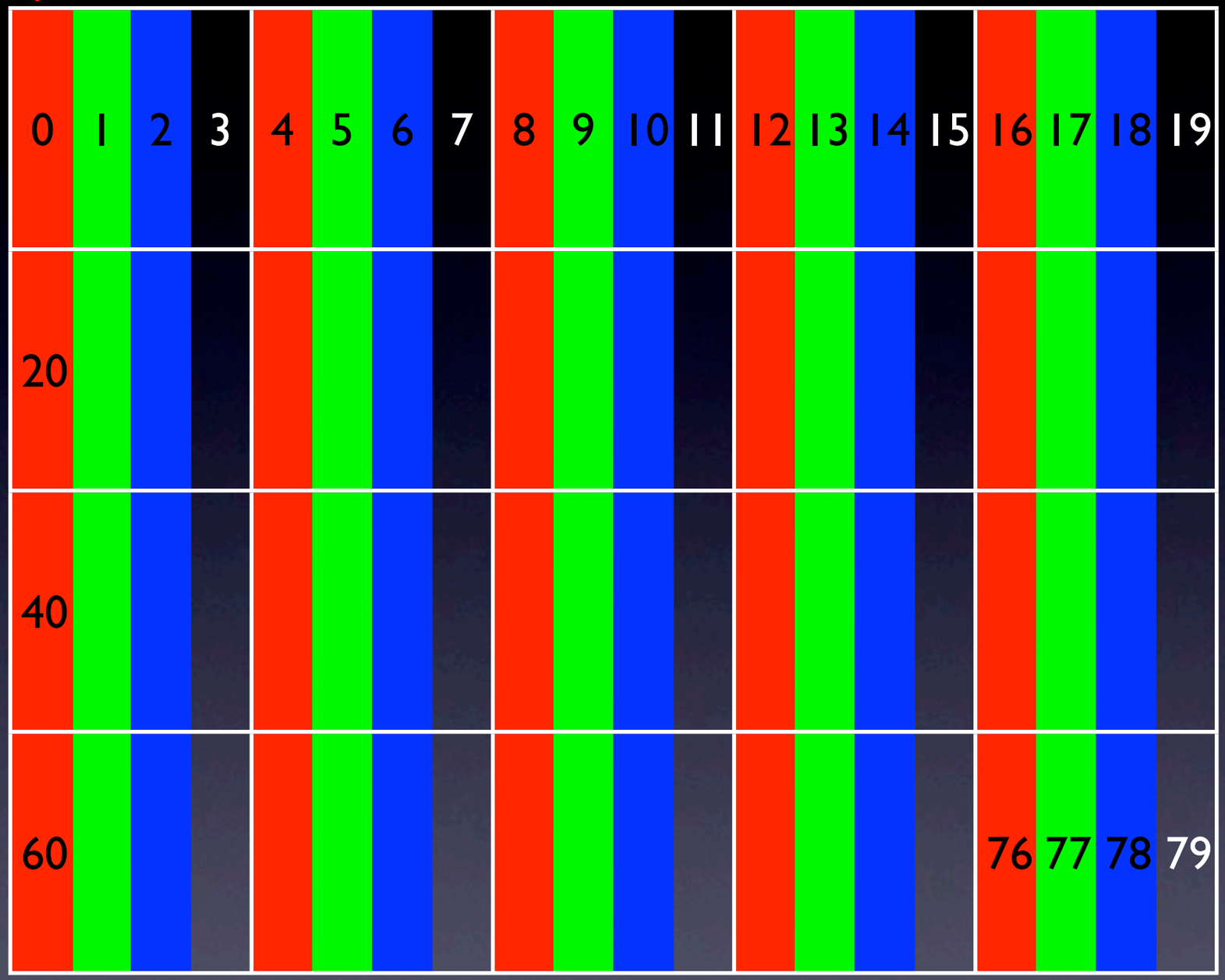
Exemple: 32 bits/pixel

- 1 octet de rouge
  1 octet de vert
  1 octet de bleu
  1 octet inutilisé
  = 2^{24} couleurs (environ 16 millions)
- Dimensions 5 x 4 pixels

adrBitmap

5 colonnes

4 lignes



adrPixel = adrBitmap + 4(ligne × largeur + colonne)

Pour simplifier les accès, on utilise une structure:

```
typedef struct
{
    UInt8 r;
    UInt8 g;
    UInt8 b;
    UInt8 x;

} PixelRGBx;
```

Avec un pointeur:

```
PixelRGBx *pixelPtr;
pixelPtr = (PixelRGBx *)bitmapPtr + ligne*largeur +
colonne;
```

Octets d'alignement

- Pour des raisons de performances, il se peut que des octets d'alignements (inutilisés) soient ajoutés en fin de ligne.

```
PixelRGBx *pixelPtr;
pixelPtr = (PixelRGBx *)bitmapPtr + ligne*(largeur +
alignement) + colonne;
```

➡Y prendre garde si on récupère une bitmap créée par le système d'exploitation.

Création

Formats gérés par CG

Color Space	Pixel format and bitmap information constant
Gray	8 bpp, 8 bpc,`kCGImageAlphaNone`
Null	8 bpp, 8 bpc, `kCGImageAlphaOnly`
RGB	16 bpp, 5 bpc, `kCGImageAlphaNoneSkipFirst`
RGB	32 bpp, 8 bpc, `kCGImageAlphaNoneSkipFirst`
RGB	32 bpp, 8 bpc, `kCGImageAlphaNoneSkipLast`
RGB	32 bpp, 8 bpc, `kCGImageAlphaPremultipliedFirst`
RGB	32 bpp, 8 bpc, `kCGImageAlphaPremultipliedLast`
CMYK	32 bpp, 8 bpc, `kCGImageAlphaNone`
Gray	32 bpp, 32 bpc, `kCGImageAlphaNone`|`kCGBitmapFloatComponents`
RGB	128 bpp, 32 bpc, `kCGImageAlphaNoneSkipLast` |`kCGBitmapFloatComponents`
RGB	128 bpp, 32 bpc, `kCGImageAlphaPremultipliedLast` |`kCGBitmapFloatComponents`
CMYK	128 bpp, 32 bpc, `kCGImageAlphaNone` |`kCGBitmapFloatComponents`
Gray	16 bpp, 16 bpc, `kCGImageAlphaNone`
RGB	64 bpp, 16 bpc, `kCGImageAlphaPremultipliedLast`
RGB	64 bpp, 16 bpc, `kCGImageAlphaNoneSkipLast`
CMYK	64 bpp, 16 bpc, `kCGImageAlphaNone`

Créer une bitmap

```
CGContextRef CGBitmapContextCreate (
    void *data,
    size_t width,
    size_t height,
    size_t bitsPerComponent,
    size_t bytesPerRow,
    CGColorSpaceRef colorspace,
    CGBitmapInfo bitmapInfo
);
```

Afficher une bitmap

1) Transformer le CGBitmapContext en CGImage:

```
CGImageRef CGBitmapContextCreateImage (
    CGContextRef c
);
```

2) L'afficher dans un CGContext:

```
void CGContextDrawImage (
    CGContextRef c,
    CGRect rect,
    CGImageRef image
);
```

Obtenir le CGContext courant

- Avec Cocoa Touch:

```
CGContextRef context
  = UIGraphicsGetCurrentContext();
```

- Avec Cocoa Mac:

```
CGContextRef context
  = [[NSGraphicsContext currentContext] graphicsPort];
```

Génération d'images

Ex. 1: Bitmap bleue

- Pour écrire vite:

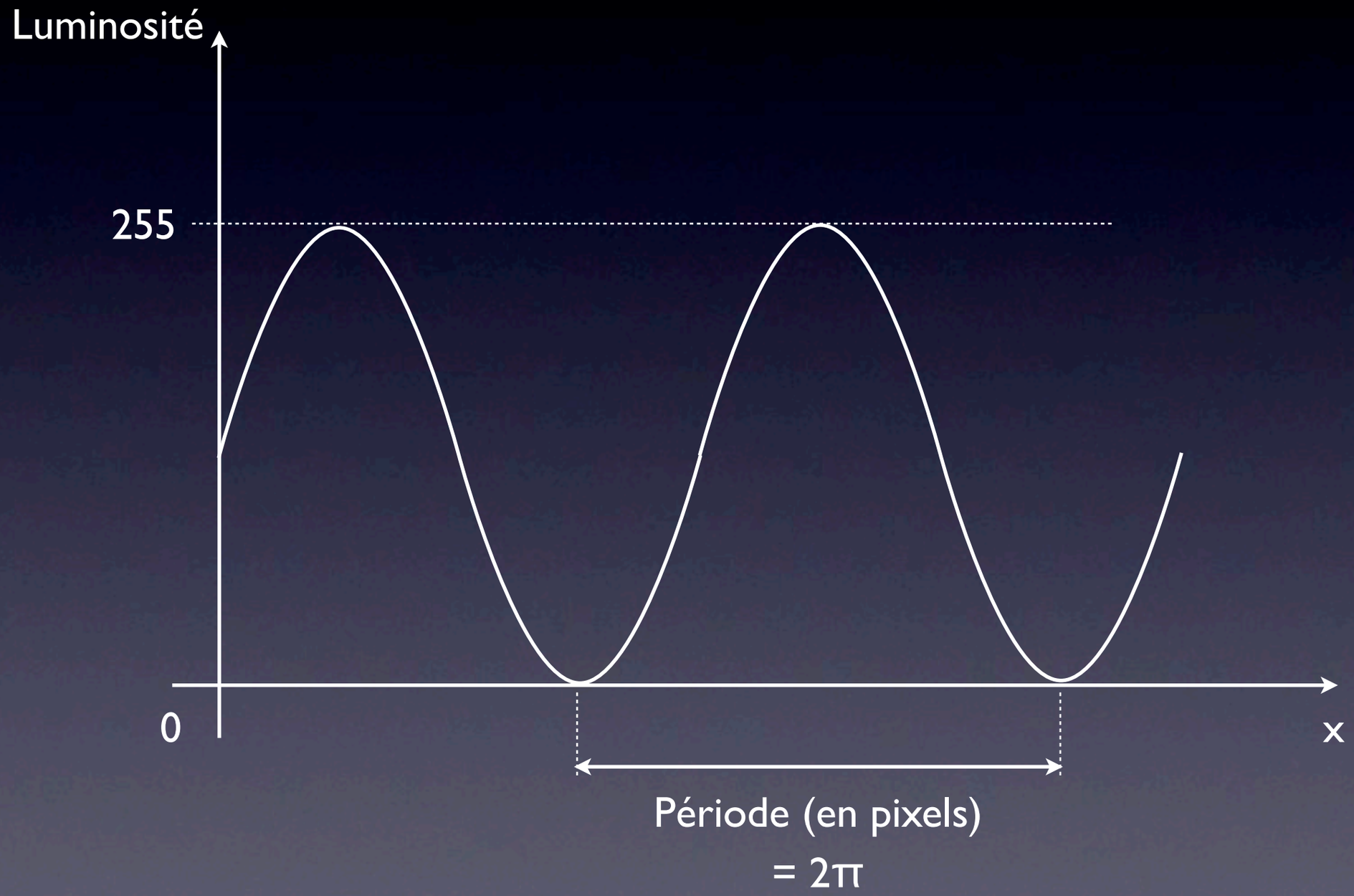
```
typedef union
{
    struct
    {
        UInt8 r;
        UInt8 g;
        UInt8 b;
        UInt8 x;

    } component;

    UInt32 all;

} PixelRGBx;
```

Ex. 2: Vaguelettes



Effets

Désaturation

- Approche naïve: simple moyenne
  gris =(R + V + B)/3
- Plus réaliste:
  luminance = 0,3 R + 0,59 V + 0,11 B

Bruit

- Principe: Générer un nombre aléatoire entre 0 et 255.
- On règle la proportion du bruit / pixel d'entrée.

Matrice de convolution

Calculer le pixel en appliquant des coefficients au pixel central et les pixels alentours.

N

-2	-1	0
-1	1	1
0	1	2

W E

S

```
pixelCourant = (
-2.pixelNW -1.pixelN +0.pixelNE
-1.pixelW  +1.pixelC +1.pixelE
+0.pixelSW +1.pixelS +2.pixelSE)
/ (-2 -1 +0 -1 +1 +1 +0 +1 +2))
```

Matrices de convolution

1	1	1
1	1	1
1	1	1

Flou

1	2	1
2	4	2
1	2	1

Flou gaussien

0	0	0
1	1	1
0	0	0

Flou horizontal

1	1	1
1	-16	1
1	1	1

Accentuation

1	1	1
1	-8	1
1	1	1

Détection de bords

2	1	0
1	1	-1
0	-1	-2

Bas-relief

Vous avez faim ?

- Les bitmaps révèlent une grande variété d'applications.
- Il s'agit de développement de bas niveau (attention aux pointeurs et aux tailles des données).
- Tous les calculs sont effectués par le CPU.

La suite…

Core Image

À bientôt !

Code source des exemples:

http://www.renaudpradenc.com/cocoaheads

Me contacter:

renaud@ceroce.com

CÉROCE