heig-vd

Mini OpenGL en C++

heig-vd

Objectifs

- Ecrire un Renderer en quelques centaines de lignes de C++
- Purement software, mais en n’oubliant pas qu’en pratique il tournerait sur la carte graphique
- Pas à pas …
- Idéalement jusqu’à arriver à l’image ci-contre

heig-vd

Format d'affichage

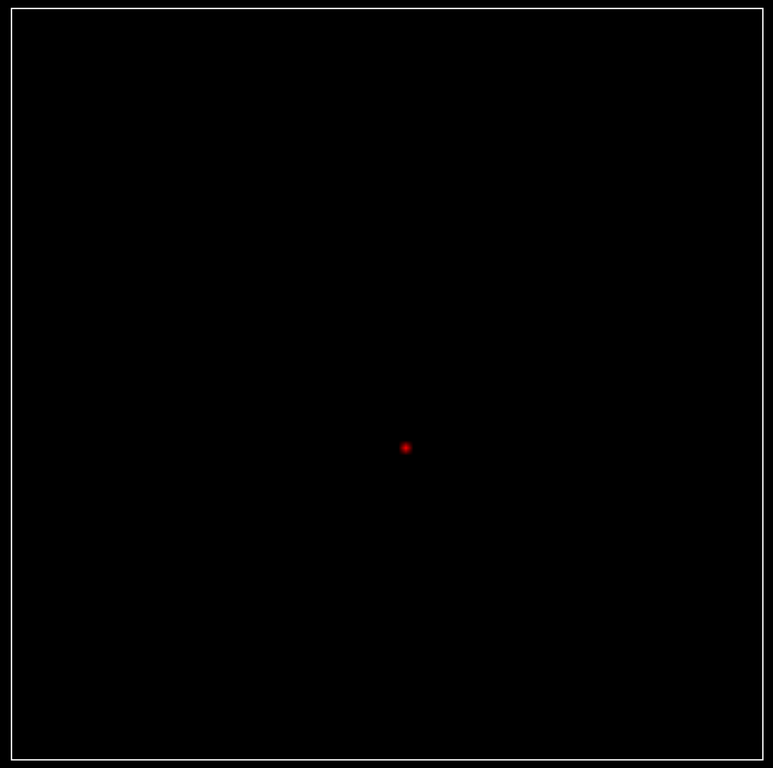
- Images au format TGA
  - Simple
  - Pixels au formats BW, RGB ou RGBA
- Je vous fourni une classe TGAImage minimaliste.
  - Constructeur définit la taille
  - La méthode set permet d'écrire dans un pixel

```
class TGAImage {
protected:
    unsigned char* data;
    int width;
    int height;
    int bytespp;

    bool   load_rle_data(std::ifstream &in);
    bool unload_rle_data(std::ofstream &out);
public:
    enum Format {
        GRAYSCALE=1, RGB=3, RGBA=4
    };

    TGAImage();
    TGAImage(int w, int h, int bpp);
    TGAImage(const TGAImage &img);
    bool read_tga_file(const char *filename);
    bool write_tga_file(const char *filename, bool rle=true);
    bool flip_horizontally();
    bool flip_vertically();
    bool scale(int w, int h);
    TGAColor get(int x, int y);
    bool set(int x, int y, TGAColor c);
    ~TGAImage();
    TGAImage & operator =(const TGAImage &img);
    int get_width();
    int get_height();
    int get_bytespp();
    unsigned char *buffer();
    void clear();
};
```

heig-vd

Exemple

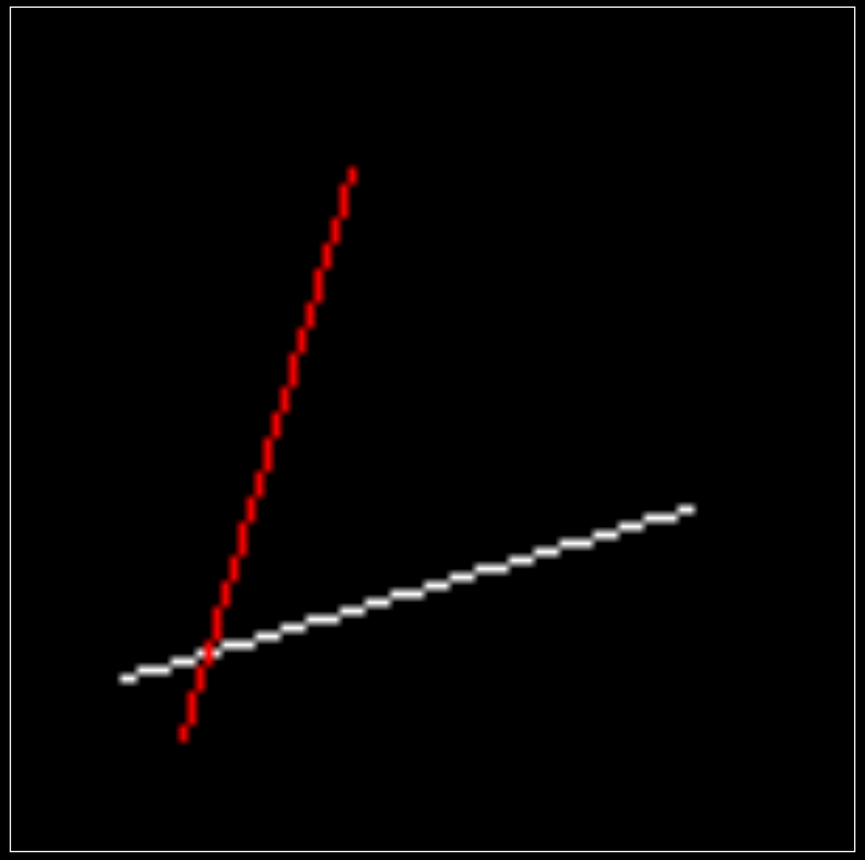
- Crée une image 100x100 noire
- Colorie le pixel 52,41 en rouge
- Sous le résultat dans output.tga



```cpp
#include "tgaimage.h"

const TGAColor white = TGAColor(255, 255, 255, 255);
const TGAColor red   = TGAColor(255, 0,   0,   255);

int main(int argc, char** argv) {

    TGAImage image(100, 100, TGAImage::RGB);
    image.set(52, 41, red);
    image.flip_vertically();
    image.write_tga_file("output.tga");
    return 0;

}
```

A vous de jouer…

- A vous d'écrire la fonction line pour que ce programme produise l'image ci-dessous…



```
#include "tgaimage.h"

const TGAColor white = TGAColor(255, 255, 255, 255);
const TGAColor red   = TGAColor(255, 0,   0,   255);

void line(int x0, int y0, int x1, int y1,
          TGAImage &image, TGAColor color);

int main(int argc, char** argv) {
    TGAImage image(100, 100, TGAImage::RGB);
    line(13, 20, 80, 40, image, white);
    line(20, 13, 40, 80, image, red);
    image.flip_vertically();
    image.write_tga_file("output.tga");
    return 0;
}
```

1er essai…

```
void line(int x0, int y0, int x1, int y1,
TGAImage &image, TGAColor color)
{
    for (float t=0.; t<1.; t+=.01) {
        int x = x0 + (x1-x0)*t;
        int y = y0 + (y1-y0)*t;
        image.set(x, y, color);
    }
}
```

- Fonctionne … apparemment
- Méchant nombre magique …

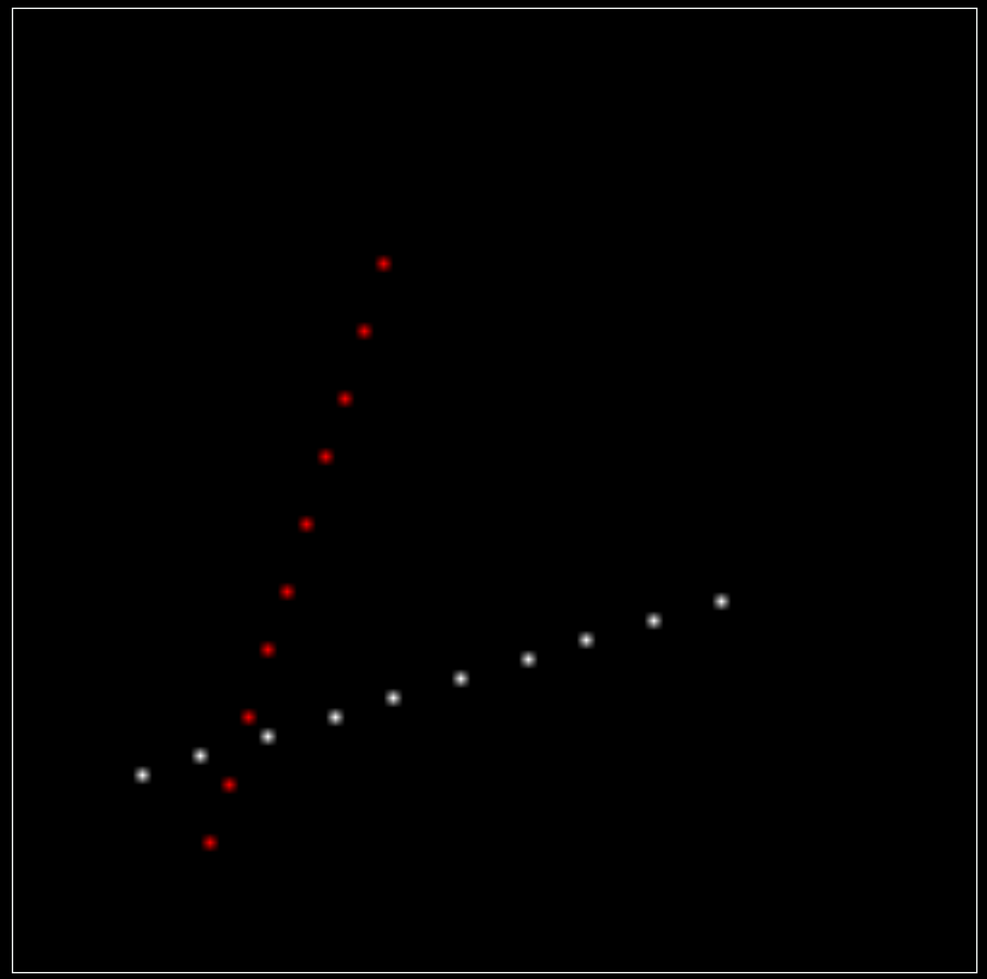
  `t+=.01`

  - Pas efficace si le pas est trop petit
  - Pas correct si le pas est trop grand

t+=.1

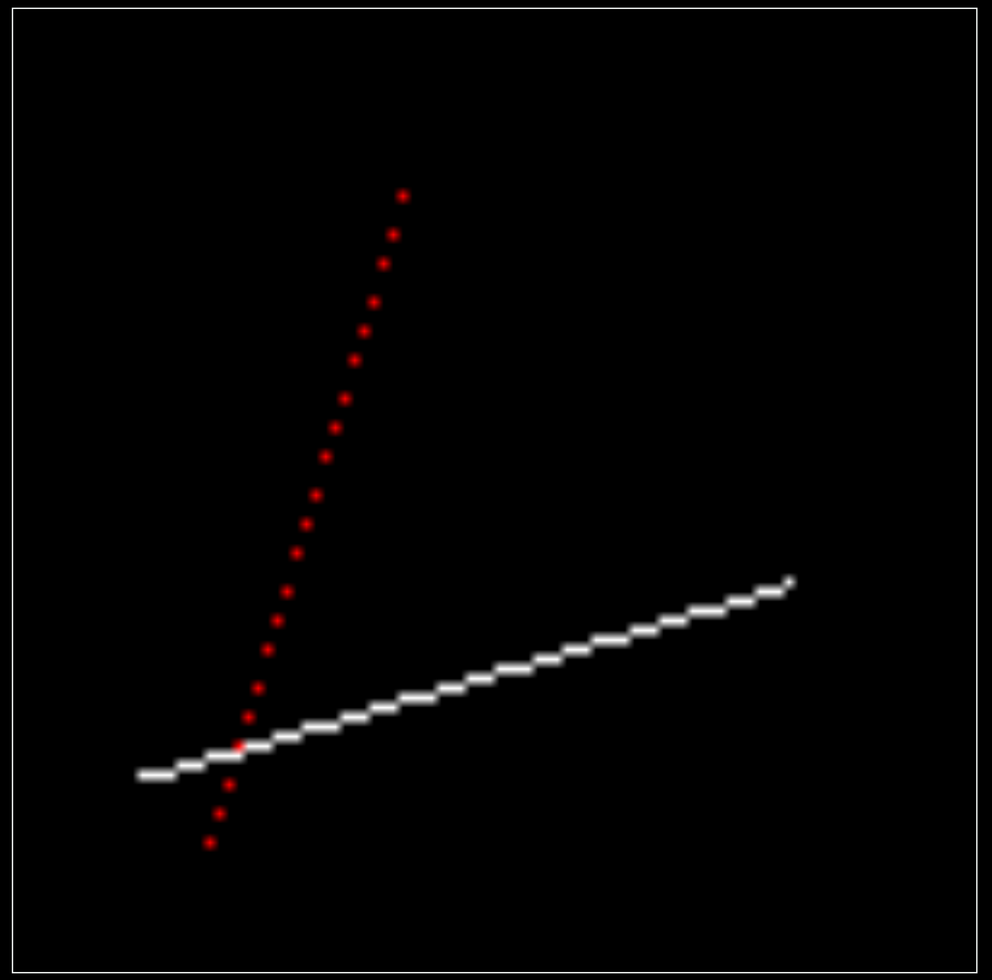
heig-vd

```
void line(int x0, int y0, int x1, int y1,
TGAImage &image, TGAColor color)
{
    for (float t=0.; t<1.; t+=.1) {
        int x = x0 + (x1-x0)*t;
        int y = y0 + (y1-y0)*t;
        image.set(x, y, color);
    }
}
```



Calculons la valeur du pas

```
void line(int x0, int y0, int x1, int y1,
          TGAImage &image, TGAColor color)
{
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        image.set(x, y, color);
    }
}
```



- Est-ce correct ?

heig-vd

Corrigeons…

- Pour les lignes penchées à plus de 45 degrés, transposons x et y
- Pour les lignes allant de droite à gauche, inversons leur sens
- C’est mieux … mais pourquoi faire autant de calculs en virgule flottante ? Et toujours diviser par x1-x0 ?

```
void line(int x0, int y0, int x1, int y1,
          TGAImage &image, TGAColor color)
{
    bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
        std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }
    if (x0>x1) {
        std::swap(x0, x1);
        std::swap(y0, y1);
    }
    for (int x=x0; x<=x1; x++) {
        float t = (x-x0)/(float)(x1-x0);
        int y = y0*(1.-t) + y1*t;
        if (steep) image.set(y, x, color);
        else       image.set(x, y, color);
    }
}
```

- Sortons la division par (x1-x0) de la boucle

```
for (int x=x0; x<=x1; x++) {
   float t = (x-x0)/(float)(x1-x0);
   int y = y0*(1.-t) + y1*t;
   if (steep) image.set(y, x, color);
   else       image.set(x, y, color);
}
```

```
float dt = 1.f/(x1-x0);
float t = 0;
for (int x=x0; x<=x1; x++) {
    t += dt;
    int y = y0*(1.-t) + y1*t;
    if (steep) image.set(y, x, color);
    else       image.set(x, y, color);
}
```

heig-vd

- Calculons y directement plutôt que de passer par la variable t

```
float dt = 1.f/(x1-x0);
float t = 0;
for (int x=x0; x<=x1; x++) {
    t += dt;
    int y = y0*(1.-t) + y1*t;
    if (steep) image.set(y, x, color);
    else       image.set(x, y, color);
}
```

```
float ystep = (y1-y0)/float(x1-x0);
float y = y0;
for (int x=x0; x<=x1; x++) {
    y += ystep;
    if (steep) image.set(int(y), x, color);
    else       image.set(x, int(y), color);
}
```

heig-vd

- Séparons les parties entières et décimales de `y = yi + sign(y1-y0) * yp`

```
float ystep = (y1-y0)/float(x1-x0);
float y = y0;
for (int x=x0; x<=x1; x++) {
    y += ystep;
    if (steep) image.set(int(y), x, color);
    else       image.set(x, int(y), color);
}
```

```
float ystep = std::abs(y1-y0)/float(x1-x0);
int dysign = (y1 > y0) ? 1 : -1;
int yi = y0;
float yp = 0; // y = yi + dysign * yp;
for (int x=x0; x<=x1; x++) {
    yp += ystep;
    if(yp > 0.5) {
        yp -= 1;
        yi += dysign;
    }
    if (steep) image.set(yi, x, color);
    else       image.set(x, yi, color);
}
```

heig-vd

- Multiplions tout pas 2(x1-x0) pour effectuer les calculs en nombres entiers

```
float ystep = std::abs(y1-y0)/float(x1-x0);
int dysign = (y1 > y0) ? 1 : -1;
int yi = y0;
float yp = 0; // y = yi + dysign * yp;
for (int x=x0; x<=x1; x++) {
    yp += ystep;
    if(yp > 0.5) {
        yp -= 1;
        yi += dysign;
    }
    if (steep) image.set(yi, x, color);
    else       image.set(x, yi, color);
}
```

```
int dx = (x1-x0);
int ystep = std::abs(2*(y1-y0));
int dysign = (y1>y0) ? +1:-1;
int yi = y0;
int yp = 0; // y = yi + dysign * float(yp)/(2*dx)
for (int x=x0; x<=x1; x++) {
    yp += ystep;
    if(yp > dx) {
        yp -= 2*dx;
        yi += dysign;
    }
    if (steep) image.set(yi, x, color);
    else       image.set(x, yi, color);
}
```

Algorithme de Bresenham

heig-vd

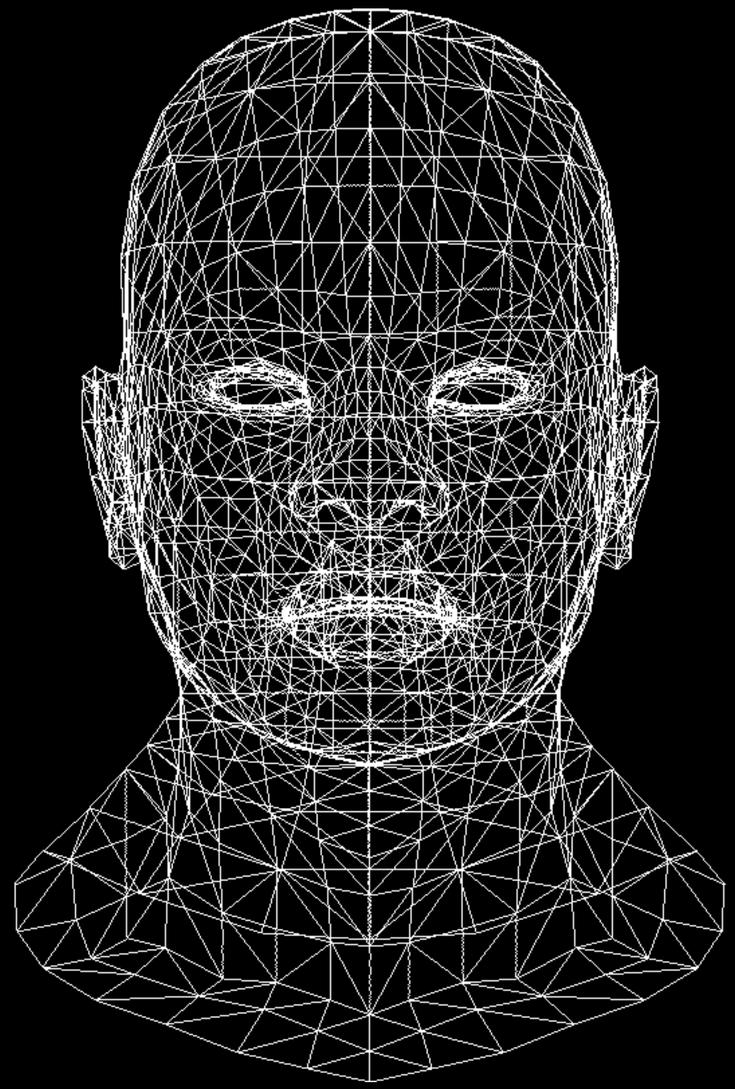
```
void line(int x0, int y0, int x1, int y1,
          TGAImage &image, TGAColor color)
{
    bool steep = false;
    if (std::abs(x0-x1)<std::abs(y0-y1)) {
       std::swap(x0, y0);
        std::swap(x1, y1);
        steep = true;
    }
    if (x0>x1) {        std::swap(x0, x1);
        std::swap(y0, y1);
    }
    int dx = (x1-x0);
    int ystep = std::abs(2*(y1-y0));
    int dysign = (y1>y0) ? +1:-1;
    int yi = y0;
    int yp = -dx; // y = yi + dysign * float(yp+dx)/(2*dx)
    for (int x=x0; x<=x1; x++) {
        yp += ystep;
        if(yp > 0) {
            yp -= 2*dx;
            yi += dysign;
        }
        if (steep) image.set(yi, x, color);
        else       image.set(x, yi, color);
    }
}
```

heig-vd

Wireframe

Dessiner une ligne …

- Le plus simple rendu 3D est le rendu filaire (wireframe rendering)
- Pour dessiner un triangle, on dessine 3 segments de ligne



heig-vd

african_head.obj

```
# List of geometric vertices, with (x, y, z [,w]) coordinates, w is
optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  …

# List of texture coordinates, in (u, [,v ,w]) coordinates, these
will vary between 0 and 1. v, w are optional and default to 0.
  vt 0.500 1 [0]
  vt ...
  …

# List of vertex normals in (x,y,z) form; normals might not be unit
vectors.
  vn 0.707 0.000 0.707
  vn ...
  …

# Parameter space vertices in ( u [,v] [,w] ) form; free form
geometry statement ( see below )
  vp 0.310000 3.210000 2.100000
  vp ...
  …

# Polygonal face element (see below)
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f 7//1 8//2 9//3
  f ...
  …

# Line element (see below)
  l 5 8 1 2 4 9
```

heig-vd

model.h

```
class Model {
private:
    std::vector<Vec3f> verts_;
    std::vector<std::vector<int> > faces_;
public:
    Model(const char *filename);
    ~Model();
    int nverts();
    int nfaces();
    Vec3f vert(int i);
    std::vector<int> face(int idx);
};
```

```
# List of geometric vertices, with (x, y, z [,w]) coordinates, w is
optional and defaults to 1.0.
  v 0.123 0.234 0.345 1.0
  v ...
  …

# List of texture coordinates, in (u, [,v ,w]) coordinates, these
will vary between 0 and 1. v, w are optional and default to 0.
  vt 0.500 1 [0]
  vt ...
  …

# List of vertex normals in (x,y,z) form; normals might not be unit
vectors.
  vn 0.707 0.000 0.707
  vn ...
  …

# Parameter space vertices in ( u [,v] [,w] ) form; free form
geometry statement ( see below )
  vp 0.310000 3.210000 2.100000
  vp ...
  …

# Polygonal face element (see below)
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f 7//1 8//2 9//3
  f ...
  …

# Line element (see below)
  l 5 8 1 2 4 9
```

Votre but

- Boucler sur tous les triangles
- Boucler sur les 3 cotés
- Boucler sur les 2 extrémités du côté
- Convertir les coordonnées 3d entre -1 et 1 en coordonnées 2d dans l’image
- Dessiner la ligne