Introduction à la programmation 3D via OpenGL

Pierre Bourdon

LSE

19 octobre 2011

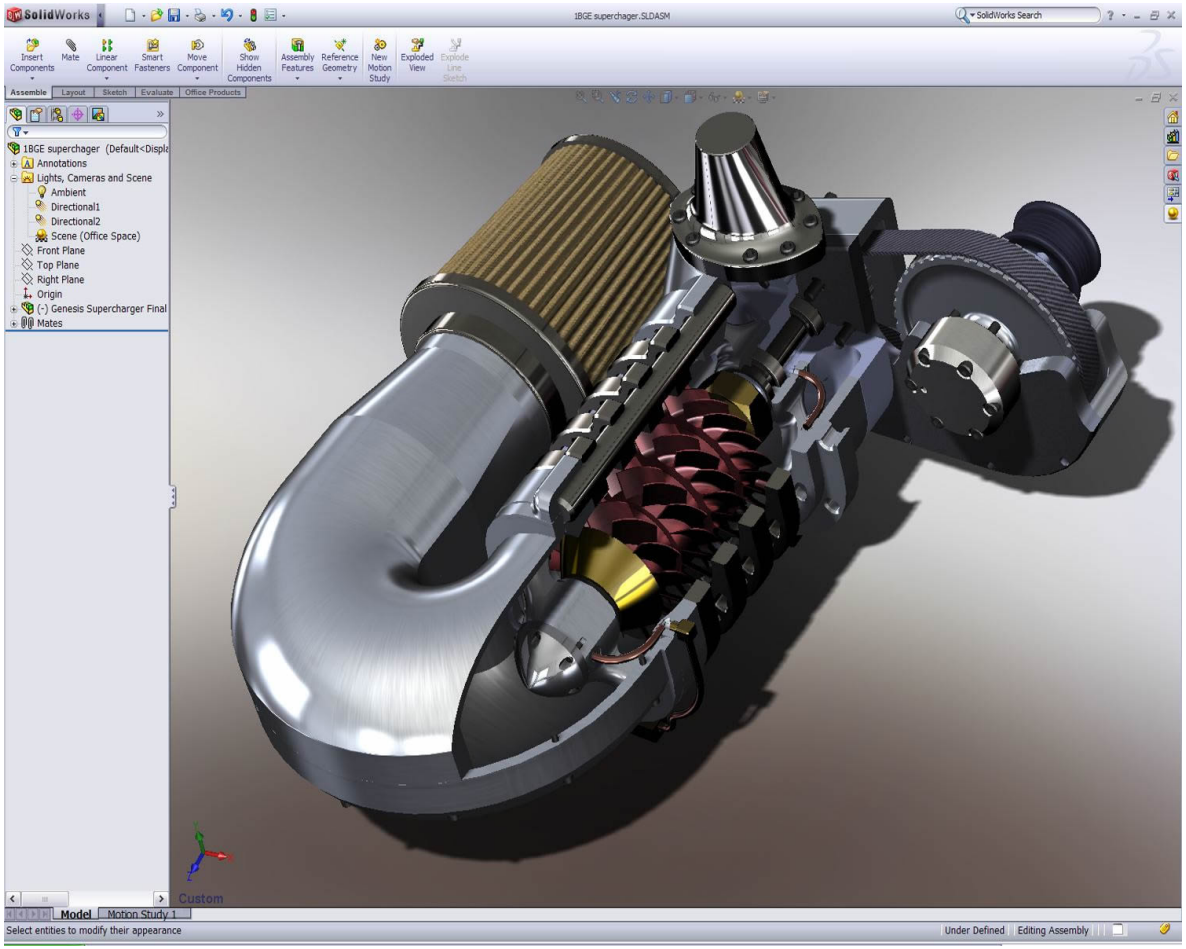
Introduction

- API permettant d'accèder à la carte graphique
- Android, iOS, Windows, Linux, Mac OS X, etc.
- Concurrent à Direct3D sous Windows
- PyOpenGL, Tao OpenGL, GLCaml, etc.

Exemples d'applications utilisant OpenGL

Exemples d'applications utilisant OpenGL

Exemples d'applications utilisant OpenGL



Plan

Scène 3D ?

- Ensemble d'objets 3D
- Ensemble de *vertices*
- (x, y, z), couleur, coordonnées de textures
- Coordonnées relatives à l'origine

Transformations

- Permettent de déplacer l'origine du repère
- Trois types majeurs : translation, rotation, mise à l'échelle
- Représentés par une matrice 4x4

Maths

- Matrice 4x4 = tableau de 16 nombres
- Coordonnées = matrice 4x1 = $(x, y, z, 1)$
- Multiplication de matrice = magie

Translation

Déplace l'origine du repère en $(x_o + T_x, y_o + T_y, z_o + T_z)$

Rotation

Tourne le repère d'un certain angle par rapport à un certain axe

Mise à l'échelle

Change la taille des axes du repère

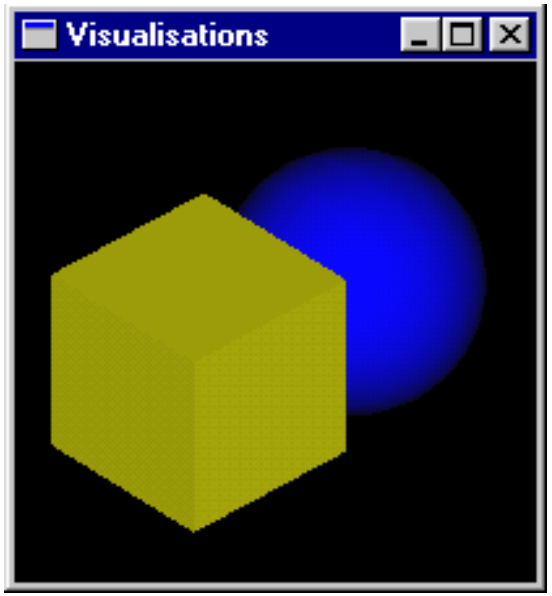
Problème

- On a une scène 3D
- On a un écran 2D
- Comment projeter la scène sur l'écran ?

Projection

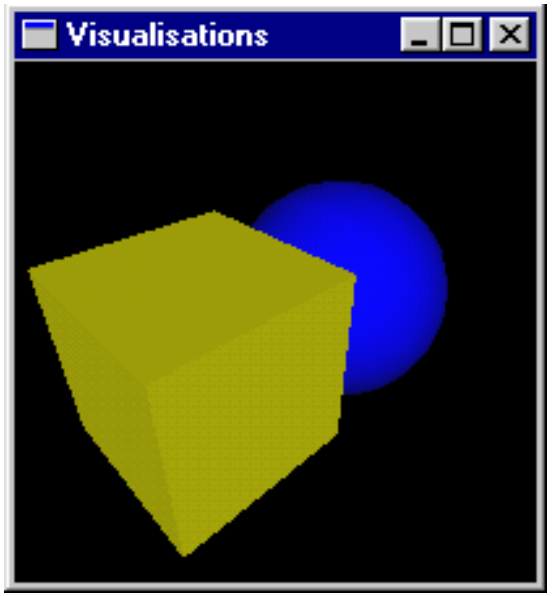
- Plusieurs moyens de projeter
- Parallèle
- En perspective

Projection parallèle



- Paramètres : left, right, bottom, top, near, far
- Décrit un parallélépipède

Projection en perspective

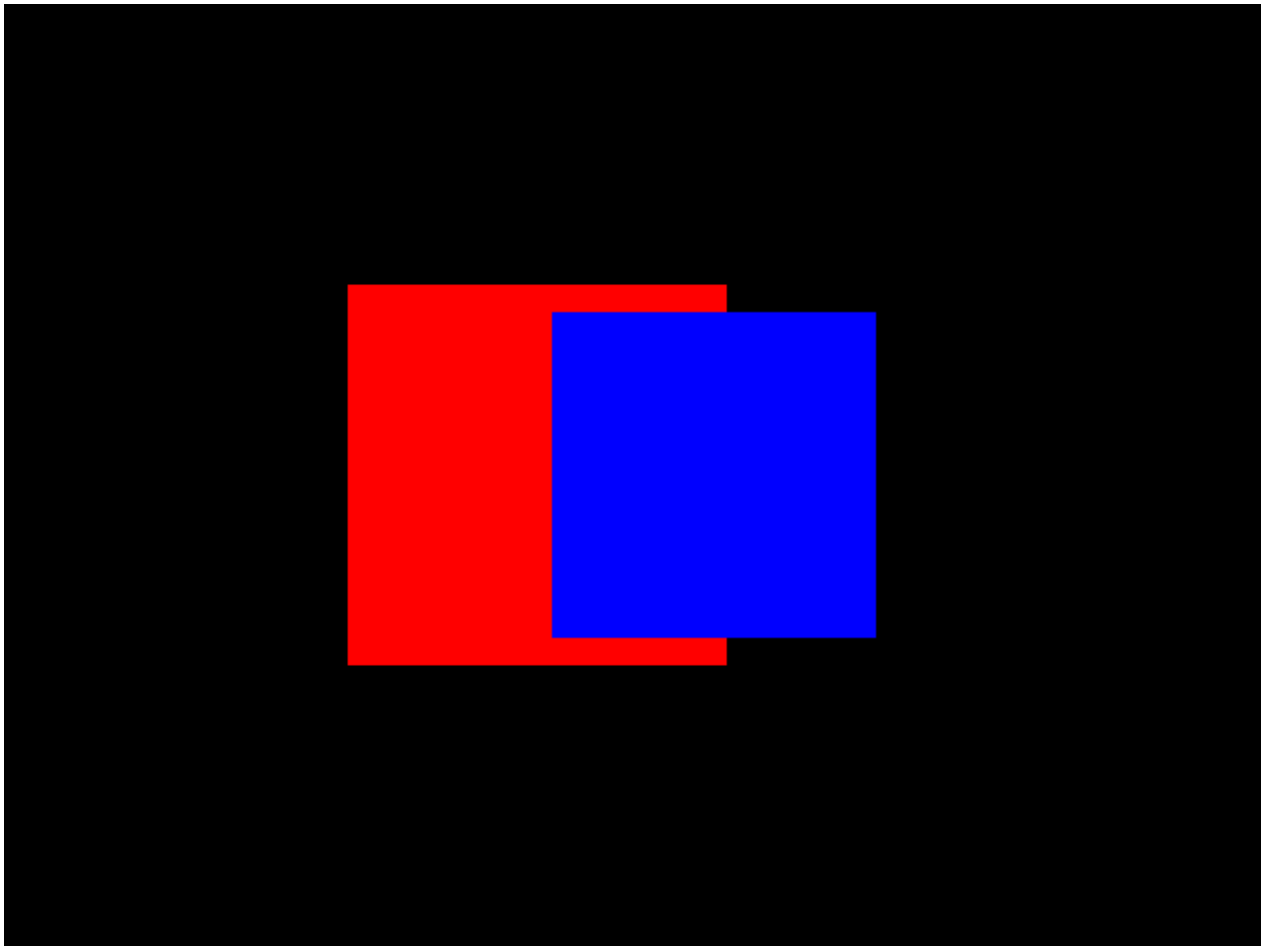


- Paramètres : FOV, aspect ratio, near, far

Problème

- On affiche un rectangle avec profondeur 0
- On affiche un rectangle avec profondeur 1 (derrière)

Problème



Solutions

- Algorithme du peintre
- Z-buffer

Algorithme du peintre

- On trie par profondeur décroissante
- On affiche de l'arrière à l'avant
- Trier demande de connaitre toutes les faces
- Problème en cas d'intersection
- Peu utilisé à cause de ces problèmes

Z-buffer

```
if depth < zbuffer[x, y]:
    zbuffer[x, y] = depth
    image[x, y] = color
else:
    # drop
```

- Grand tableau qui contient la profondeur
- Depth test
- Solution utilisée par presque tout le monde

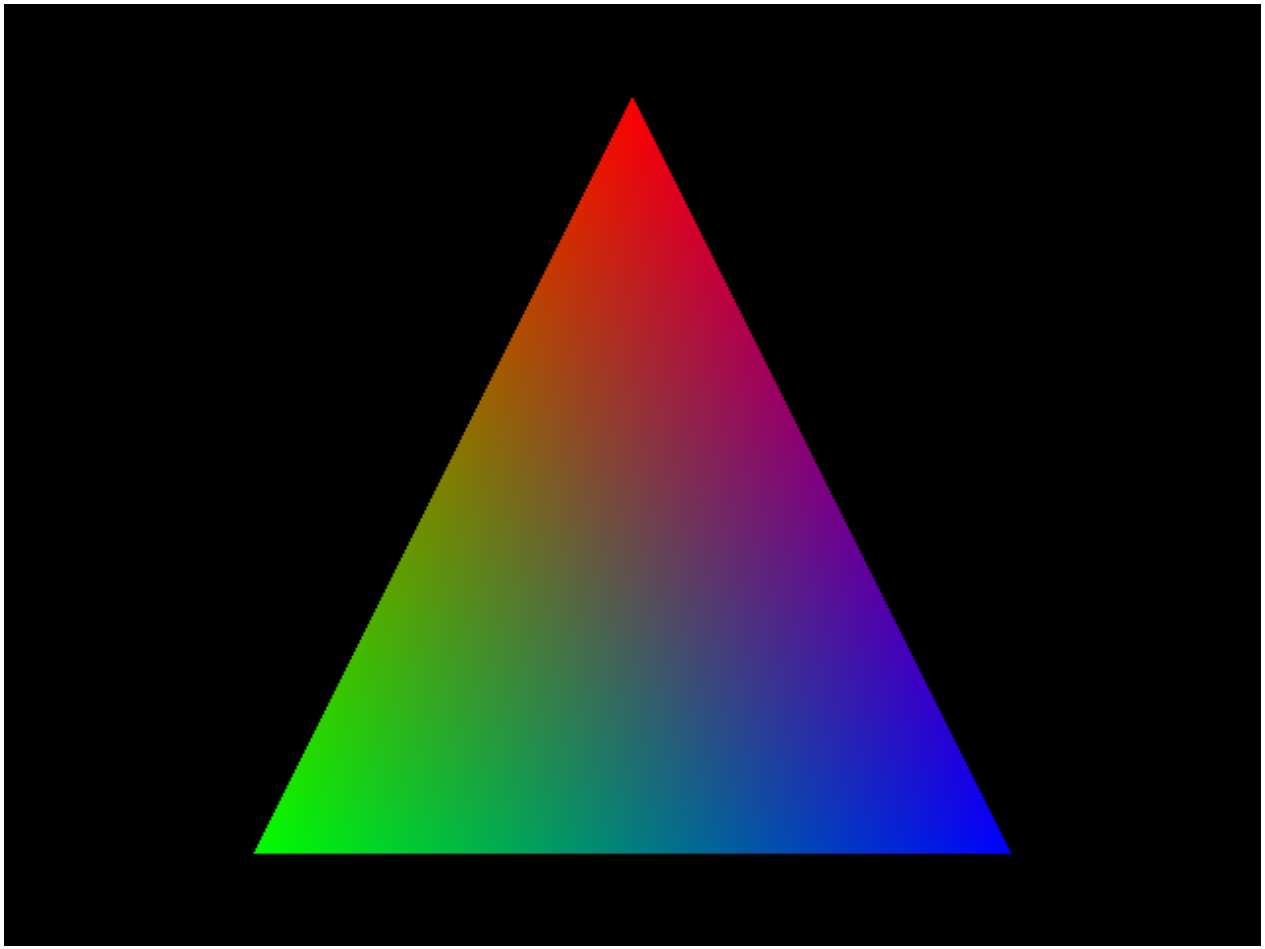
Représentation

- 4 composantes : RGBA
- Chaque composante entre 0 et 255 ou 0.0 et 1.0

Canal alpha

- Sert à représenter la transparence
- Généralement 1.0 = opaque, 0.0 = transparent

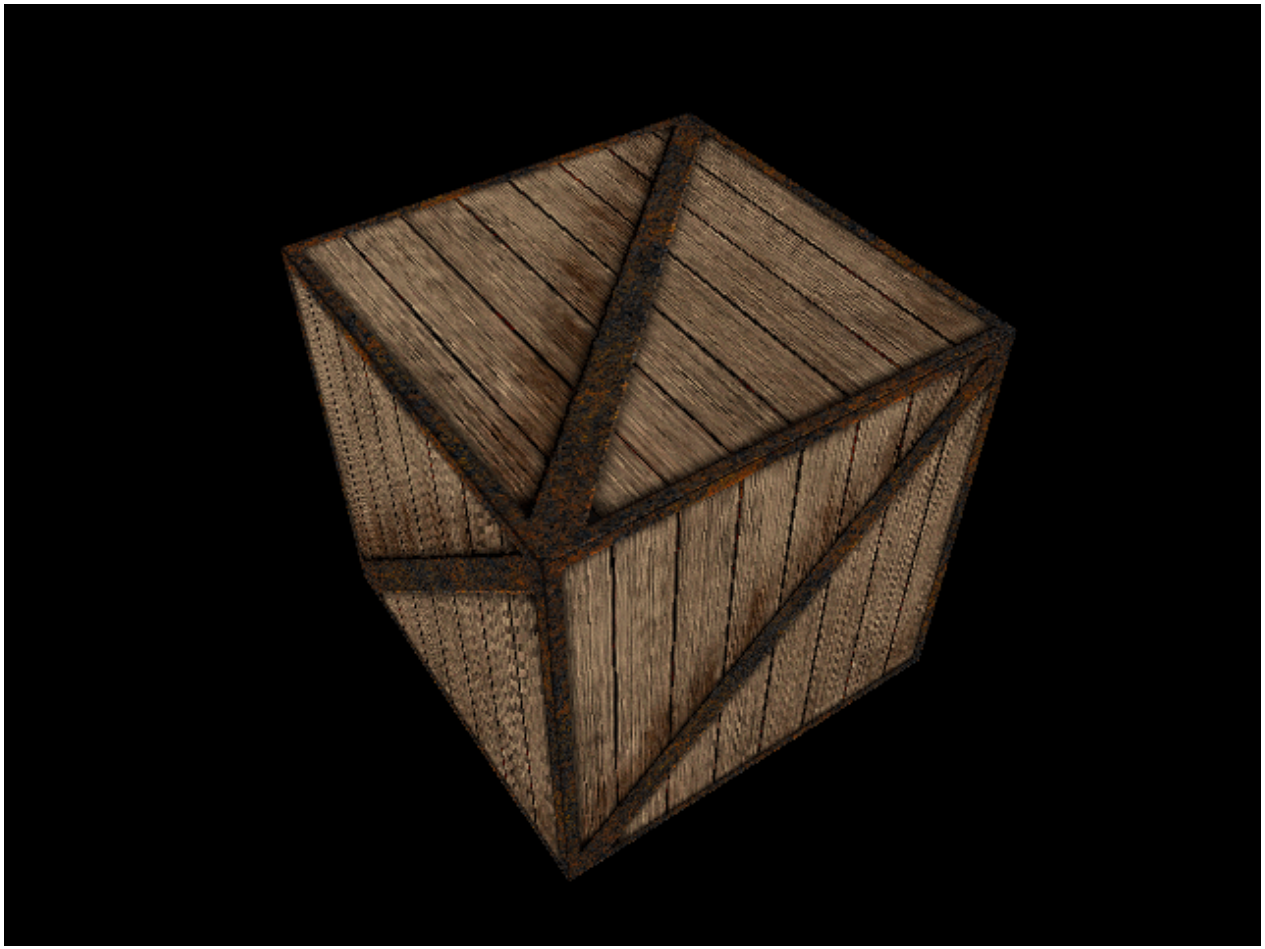
Interpolation



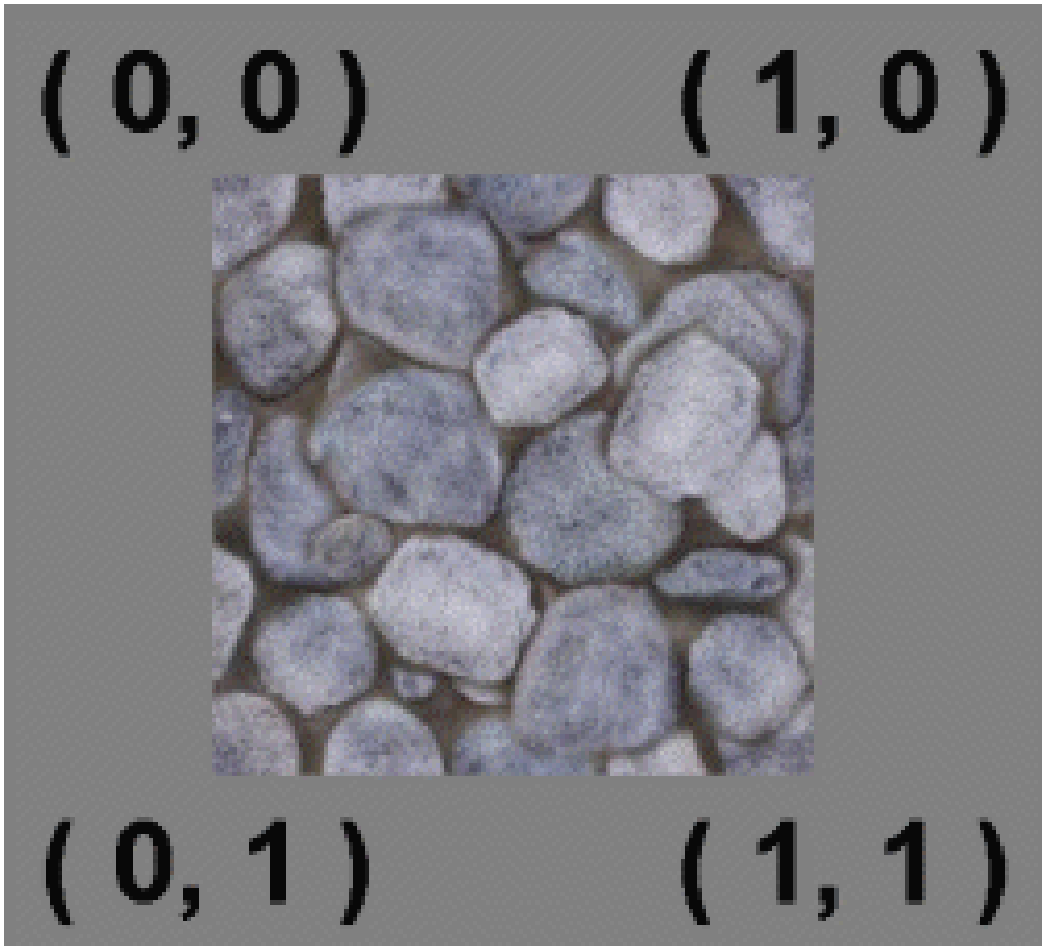
C'est quoi ?

- Une image
- Appliquée sur les faces d'un objet 3D

Exemple



Coordonnées de texture



Coordonnées de texture

- Sert à définir quel bout de la texture est sur quelle face
- Propriété d'un vertex

Exemple

Pourquoi ?

- OpenGL ne gère que l'affichage
- Pas de fenêtrage
- Pas d'événements
- Pas de son

SDL

- Simple DirectMedia Layer
- Crée une fenêtre
- Clavier/souris/joystick
- Son, musique avec SDL_mixer
- Images avec SDL_image
- Pygame, TaoSDL, OCamlSDL, etc.

Initialisation de SDL

```c
#include <SDL/SDL.h>
#include <stdlib.h>

int main(void)
{
    SDL_Init(SDL_INIT_EVERYTHING);
    atexit(SDL_Quit);

    /* ... */

    return 0;
}
```

Création d'une fenêtre

```
SDL_SetVideoMode(800, /* width */
                 600, /* height */
                 0, /* bpp, 0 = auto */
                 SDL_OPENGL); /* flags */
```

Création d'une fenêtre fullscreen

```
SDL_SetVideoMode(800, /* width */
                 600, /* height */
                 0, /* bpp, 0 = auto */
                 SDL_OPENGL | SDL_FULLSCREEN); /* flags
```

Boucle principale

```
while (!quit)
{
    handle_events();
    update_game_state();
    display_with_gl();

    SDL_GL_SwapBuffers();
}
```

Gestion des événements

```
SDL_Event evt;
while (SDL_PollEvent(&evt))
{
    if (evt.type == SDL_KEYDOWN) /* ... */
    else if (evt.type == SDL_MOUSEMOTION) /* ... */
    else if (evt.type == SDL_MOUSEBUTTONDOWN) /* ... */
    else if (evt.type == SDL_QUIT) /* ... */
}
```

... et c'est tout !

- On n'ira pas plus loin
- Tonnes de tutoriels en ligne
- C'est une conférence OpenGL FFS !

Initialisation

```
glClear(GL_COLOR_BUFFER_BIT);
/* ou */
glClear(GL_DEPTH_BUFFER_BIT);
```

Afficher un triangle en 2D

```
glBegin(GL_TRIANGLES);
    glVertex2f(0.0, 0.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(0.5, 1.0);
glEnd();
```

Afficher un carré en 2D

```
glBegin(GL_TRIANGLES);
    glVertex2f(0.0, 0.0); glVertex2f(0.0, 1.0);
        glVertex2f(1.0, 0.0);

    glVertex2f(1.0, 1.0); glVertex2f(0.0, 1.0);
        glVertex2f(1.0, 0.0);
glEnd();

glBegin(GL_QUADS);
    glVertex2f(0.0, 0.0);
    glVertex2f(1.0, 0.0);
    glVertex2f(1.0, 1.0);
    glVertex2f(0.0, 1.0);
glEnd();
```

Donner une couleur

```
glColor4ub(255, 0, 0, 255); /* r, g, b, a */
glVertex2f(0.0, 0.0);

glColor4f(0.0, 1.0, 0.0, 1.0);
glVertex2f(0.0, 1.0);
```

3D ?

- Même chose mais avec 3f au lieu de 2f
- glEnable(GL_DEPTH_TEST) pour avoir un zbuffer
- Pour avoir un résultat correct -> projection

Types de matrices

- glMatrixMode
- Matrice de projection (GL_PROJECTION)
- Matrice de modèle (GL_MODELVIEW)
- Autres types dont on ne parlera pas

Pile de matrices

- glPushMatrix / glPopMatrix
- Permet de facilement annuler une transformation

Mise en place de la projection

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(70, /* fov */
               (float)width / height, /* aspect ratio */
               0.1, /* near */
               10000.0); /* far */
```

Mise en place de la projection

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(-10.0, 10.0, /* xmin, xmax */
        -10.0, 10.0, /* ymin, ymax */
        -10.0, 10.0); /* zmin, zmax */
```

Transformations

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslate3f(1.0, 0.0, 1.0);
glScale3f(1.0, 2.0, 1.5);
glRotatef(45, 1.0, 0.0, 0.0);
```

glEnable / glDisable

- Plein de choses optionnelles
- glEnable active
- glDisable désactive
- glIsEnabled -> actif ou pas

Exemples de paramètres

- GL_DEPTH_TEST : zbuffer
- GL_TEXTURE_2D : textures
- GL_LIGHTING : éclairage
- GL_BLEND : gère la transparence

Paramètres plus spécifiques

- glBlend(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
- glDepthFunc(GL_LEQUAL)
- ...

Comment ça marche ?

- On demande à OpenGL de nous générer une texture
- On lui envoie l'image correspondante
- Ensuite, avant d'afficher, on *bind* la texture

Différents formats

- RGBA
- RGB sans alpha
- Greyscale
- Textures compressées

En code : génération

```
int tex_id;
glGenTextures(1, &tex_id);

/* envoyer l'image */
```

En code : afficher avec une texture

```
glBindTexture(GL_TEXTURE_2D, tex_id);
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex2f(0.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex2f(1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex2f(1.0, 1.0);
    glTexCoord2f(0.0, 1.0); glVertex2f(0.0, 1.0);
glEnd();
```

Charger une texture

- C'est compliqué
- Plein de tutos sur le net
- Dépend du format d'image
- Regardez du côté de SOIL

free()

```
glDeleteTextures(1, &tex_id);
```

Display lists

- Enregistre une liste de commande GL
- Permet d'appeller toute la liste en une fois
- Commandes limitées (pas de bind)
- Simple et plutôt efficace

Vertex Array Object

- Tableau de vertices
- Au lieu d'envoyer un par un, on envoie le tableau entier
- Beaucoup plus rapide
- Vertices renvoyés à la carte graphique à chaque rendu

Vertex Buffer Object

- Tableau de vertices aussi
- Contrairement aux VAO, on contrôle l'envoi à la carte graphique
- Très utile pour des géométries statiques

Index Buffer Object

- Permet d'éviter de dupliquer les vertices
- Référence les vertices par un numéro
- Très utile en combinaison avec les VBO et VAO

Frame Buffer Object

- Récupère le rendu OpenGL dans une texture
- Permet de faire des effets comme des portails ou des miroirs
- Très facile à mettre en place
- Ralentit pas mal le rendu

Vertex shader

- Permet de faire tourner du code pour chaque vertex traité
- Modifier couleur, profondeur, coordonnées de textures, etc.
- Se programme en GLSL
- Extrémement utile, extrémement utilisé

Fragment shader

- Aussi appellé Pixel Shader
- Permet de faire tourner du code pour chaque pixel traité
- Modifier couleur, profondeur, etc.
- Se programme aussi en GLSL
- Pratiquement tous les effets modernes passent par là

Des questions ?

- Merci de votre attention !
- #epita @ irc.rezosup.org