OpenGL – les bases

David Odin

Septembre 2007

OpenGL – Plan

OpenGL – Plan

Présentation générale

- Proposé par SGI

Présentation générale

- Proposé par SGI
- Dérivé de IrixGL suivant les besoins du moment

Présentation générale

- Proposé par SGI
- Dérivé de IrixGL suivant les besoins du moment
- Très lié aux possibilités en électronique de l'époque

Présentation générale

- Proposé par SGI
- Dérivé de IrixGL suivant les besoins du moment
- Très lié aux possibilités en électronique de l'époque
- Évolution actuelle suivant les demandes des constructeurs de cartes (et des joueurs. . .)

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3
- 2002 OpenGL 1.4

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3
- 2002 OpenGL 1.4
- 2003 OpenGL 1.5

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3
- 2002 OpenGL 1.4
- 2003 OpenGL 1.5
- 2005 OpenGL 2.0

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3
- 2002 OpenGL 1.4
- 2003 OpenGL 1.5
- 2005 OpenGL 2.0
- Aout 2006 OpenGL 2.1

HISTORIQUE

OpenGL évolue en fonction des extensions que les constructeurs rendent populaires

- Juin 1992 OpenGL 1.0
- Janvier 1996 OpenGL 1.1
- 1998 OpenGL 1.2 et 1.2.1
- Août 2001 OpenGL 1.3
- 2002 OpenGL 1.4
- 2003 OpenGL 1.5
- 2005 OpenGL 2.0
- Aout 2006 OpenGL 2.1
- Septembre 2007 OpenGL 3.0 ?

NOUVEAUTÉES D'OPENGL 1.1

Copy texture and subtexture	GL_EXT_copy_texture et GL_EXT_subtexture
Logical operation	GL_EXT_blend_logic_op
Polygon offset	GL_EXT_polygon_offset
Texture image formats	GL_EXT_texture
Texture objects	GL_EXT_texture_object
Texture proxies	GL_EXT_texture
Texture replace environment	GL_EXT_texture
Vertex array	GL_EXT_vertex_array

NOUVEAUTÉS D'OPENGL 1.2

BGRA pixel formats	GL_EXT_bgra
Imaging subset (optional)	GL_SGI_color_table, GL_EXT_color_subtable, ...
Normal rescaling	GL_EXT_rescale_normal
Packed pixel formats	GL_EXT_packed_pixels
Separate specular color	GL_EXT_separate_specular_color
Texture coordinate edge clamping	GL_SGIS_texture_edge_clamp
Texture level of detail control	GL_SGIS_texture_lod
Three-dimensional texturing	GL_EXT_texture3D
Vertex array draw element range	GL_EXT_draw_range_elements

NOUVEAUTÉS D'OPENGL 1.3

Compressed textures	GL_ARB_texture_compression
Cube map textures	GL_ARB_texture_cube_map
Multisample	GL_ARB_multisample
Multitexture	GL_ARB_multitexture
Texture add environment mode	GL_ARB_texture_env_add
Texture border clamp	GL_ARB_texture_border_clamp
Texture combine environment mode	GL_ARB_texture_env_combine
Texture dot3 environment mode	GL_ARB_texture_env_dot3
Transpose matrix	GL_ARB_transpose_matrix

NOUVEAUTÉS D'OPENGL 1.4

Automatic mipmap generation	GL_SGIS_generate_mipmap
Blend function separate	GL_ARB_blend_func_separate
Blend squaring	GL_NV_blend_square
Depth textures	GL_ARB_depth_texture
Fog coordinate	GL_EXT_fog_coord
Multiple draw arrays	GL_EXT_multi_draw_arrays
Point parameters	GL_ARB_point_parameters
Secondary color	GL_EXT_secondary_color
Separate blend functions	GL_EXT_blend_func_separate et GL_EXT_blend_color
Shadows	GL_ARB_shadow
Stencil wrap	GL_EXT_stencil_wrap
Texture crossbar environment mode	GL_ARB_texture_env_crossbar
Texture level of detail bias	GL_EXT_texture_lod_bias
Texture mirrored repeat	GL_ARB_texture_mirrored_repeat
Window raster position	GL_ARB_window_pos

NOUVEAUTÉS D'OPENGL 1.5

Buffer objects	GL_ARB_vertex_buffer_object
Occlusion queries	GL_ARB_occlusion_query
Shadow functions	GL_EXT_shadow_funcs

NOUVEAUTÉS D'OPENGL 2.0

Multiple render targets	GL_ARB_draw_buffers
Non-power-of-two textures	GL_ARB_texture_non_power_of_two
Point sprites	GL_ARB_point_sprite
Separate blend equation	GL_EXT_blend_equation_separate
Separate stencil	GL_ATI_separate_stencil et GL_EXT_stencil_two_side
Shading language	GL_ARB_shading_language_100
Shader objects	GL_ARB_shader_objects
Shader programs	GL_ARB_fragment_shader et GL_ARB_vertex_shader

OpenGL – Plan

Machine d'état

On ne le dira jamais assez : **OpenGL est une machine d'état**. Tout reste dans un état tant qu'on ne le change pas.

- Chaque réglage reste dans son état,

Machine d'état

On ne le dira jamais assez : **OpenGL est une machine d'état**. Tout reste dans un état tant qu'on ne le change pas.

- Chaque réglage reste dans son état,
- Il y a une notion de "contexte courant" : couleur, normale, etc.

Machine d'état

On ne le dira jamais assez : **OpenGL est une machine d'état**. Tout reste dans un état tant qu'on ne le change pas.

- Chaque réglage reste dans son état,
- Il y a une notion de "contexte courant" : couleur, normale, etc.
- Chaque tampon continue de contenir ce qu'il avait tant qu'on ne le vide pas, etc.

FONCTIONS PARTICULIÈRES

- Les fonctions glEnable et glDisable permettent de changer un état particulier (en général, une fonctionnalité particulière d'OpenGL)

Fonctions particulières

- Les fonctions glEnable et glDisable permettent de changer un état particulier (en général, une fonctionnalité particulière d'OpenGL)
- Les fonctions glGet* permettent de récupérer un état, ou des informations plus conséquentes comme les constituants d'une matrice, ou une chaîne de caractères décrivant un état donné.

Fonctions particulières

- Les fonctions glEnable et glDisable permettent de changer un état particulier (en général, une fonctionnalité particulière d'OpenGL)
- Les fonctions glGet* permettent de récupérer un état, ou des informations plus conséquentes comme les constituants d'une matrice, ou une chaîne de caractères décrivant un état donné.
- En particulier, la plupart des fonctions OpenGL peut positionner une erreur que l'on peut connaître en utilisant la fonction glGetError.

OpenGL – Plan

Conventions générales

- Les fonctions commencent par gl avec une majuscule à chaque mot : glBegin, glTexParameter, glViewport, etc.

Conventions générales

- Les fonctions commencent par gl avec une majuscule à chaque mot : glBegin, glTexParameter, glViewport, etc.
- Les constantes commencent par GL, sont entièrement en majuscules, les mots sont séparés par des soulignés : GL_TRIANGLES, GL_TEXTURE_2D

Conventions générales

- Les fonctions commencent par gl avec une majuscule à chaque mot : glBegin, glTexParameter, glViewport, etc.
- Les constantes commencent par GL, sont entièrement en majuscules, les mots sont séparés par des soulignés : GL_TRIANGLES, GL_TEXTURE_2D
- Pour des raisons de portabilité, OpenGL redéfinit les types de base en ajoutant le prefixe GL : GLfloat, GLint, GLdouble, etc.

Conventions générales

- Les fonctions commencent par gl avec une majuscule à chaque mot : glBegin, glTexParameter, glViewport, etc.
- Les constantes commencent par GL, sont entièrement en majuscules, les mots sont séparés par des soulignés : GL_TRIANGLES, GL_TEXTURE_2D
- Pour des raisons de portabilité, OpenGL redéfinit les types de base en ajoutant le prefixe GL : GLfloat, GLint, GLdouble, etc.
- Certaines fonctions existent en plusieurs exemplaires, dans ce cas, le nombre et/ou le type des arguments est sont donnés à la fin du nom de la fonction : glColor4f, glVertex2i, glNormal3s, etc.

Conventions générales

- Les fonctions commencent par gl avec une majuscule à chaque mot : glBegin, glTexParameter, glViewport, etc.
- Les constantes commencent par GL, sont entièrement en majuscules, les mots sont séparés par des soulignés : GL_TRIANGLES, GL_TEXTURE_2D
- Pour des raisons de portabilité, OpenGL redéfinit les types de base en ajoutant le prefixe GL : GLfloat, GLint, GLdouble, etc.
- Certaines fonctions existent en plusieurs exemplaires, dans ce cas, le nombre et/ou le type des arguments est sont donnés à la fin du nom de la fonction : glColor4f, glVertex2i, glNormal3s, etc.
- Un **v** à la fin d'un nom de fonction signifie que les paramètres seront passés sous la forme d'un tableau (ou d'un pointeur) : void glVertex3fv (GLfloat *points);

La “programmation OpenGL” consiste en l’utilisation des fonctions OpenGL.
Celles-ci permettent de faire différentes choses :

- Positionner ou changer un état (fonctionnalité, matrice courante, paramètre particulier). Il s’agit de 90 % des fonctions.

La “programmation OpenGL” consiste en l’utilisation des fonctions OpenGL.
Celles-ci permettent de faire différentes choses :

- Positionner ou changer un état (fonctionnalité, matrice courante, paramètre particulier). Il s’agit de 90 % des fonctions.
- Envoyer de la géométrie (des sommets) dans le pipeline OpenGL

La programmation OpenGL

La “programmation OpenGL” consiste en l’utilisation des fonctions OpenGL.
Celles-ci permettent de faire différentes choses :

- Positionner ou changer un état (fonctionnalité, matrice courante, paramètre particulier). Il s’agit de 90 % des fonctions.
- Envoyer de la géométrie (des sommets) dans le pipeline OpenGL
- Envoyer des images vers ou depuis la mémoire vidéo.

OpenGL – Plan

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures
- Définition des matrices de projections

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures
- Définition des matrices de projections
- La triangulation de certains objets

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures
- Définition des matrices de projections
- La triangulation de certains objets
- La création d'objets de type quadrique

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures
- Définition des matrices de projections
- La triangulation de certains objets
- La création d'objets de type quadrique
- La création de NURBS

GLU ET GLUT

OpenGL s'utilise rarement seul. Deux bibliothèques de fonctions viennent s'ajouter aux fonctionnalités OpenGL pour en rendre la programmation plus simple.
GLU (Open**GL U**tilitaire) propose un ensemble de fonction qui permet de se simplifier la vie sur plusieurs points :

- Génération des textures
- Définition des matrices de projections
- La triangulation de certains objets
- La création d'objets de type quadrique
- La création de NURBS
- Une gestion simplifiée des erreurs.

GLU ET GLUT

La bibliothèque GLUT (open**GL** **U**tility **T**oolkit) est plus orientée vers les interactions avec l'utilisateur.
Elle permet de :

- Initialiser facilement L'ensemble du système.

GLU ET GLUT

La bibliothèque GLUT (open**GL** **U**tility **T**oolkit) est plus orientée vers les interactions avec l'utilisateur.
Elle permet de :

- Initialiser facilement L'ensemble du système.
- Créer des fenêtres et des menus

La bibliothèque GLUT (open**GL** **U**tility **T**oolkit) est plus orientée vers les interactions avec l'utilisateur.
Elle permet de :

- Initialiser facilement L'ensemble du système.
- Créer des fenêtres et des menus
- Associer des fonctions (callback) à des événements. Par exemple une fonction pourra être automatiquement rappelée lorsque l'utilisateur bouge la souris ou appuie sur une touche du clavier.

GLU ET GLUT

La bibliothèque GLUT (open**GL** **U**tility **T**oolkit) est plus orientée vers les interactions avec l'utilisateur.
Elle permet de :

- Initialiser facilement L'ensemble du système.
- Créer des fenêtres et des menus
- Associer des fonctions (callback) à des événements. Par exemple une fonction pourra être automatiquement rappelée lorsque l'utilisateur bouge la souris ou appuie sur une touche du clavier.
- Afficher des caractères à l'écran (malheureusement très très limité).

GLU ET GLUT

La bibliothèque GLUT (open**GL** **U**tility **T**oolkit) est plus orientée vers les interactions avec l'utilisateur.
Elle permet de :

- Initialiser facilement L'ensemble du système.
- Créer des fenêtres et des menus
- Associer des fonctions (callback) à des événements. Par exemple une fonction pourra être automatiquement rappelée lorsque l'utilisateur bouge la souris ou appuie sur une touche du clavier.
- Afficher des caractères à l'écran (malheureusement très très limité).
- Créer des objets classiques : sphère, cube, cone, tore, polyèdre régulier, théière, etc.

OpenGL – Plan

PARTIE PRINCIPALE

```c
/* deux_quads.c */
#include <math.h>
#include <GL/glut.h>

float angle = 0.0;

int main(int argc, char** argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB); // GLUT_DEPTH, GLUT_STENCIL...
  glutInitWindowSize(250, 250);
  glutInitWindowPosition (100, 100);
  glutCreateWindow("deux_quads");
  init();
  glutDisplayFunc(display);
  glutSpecialFunc(gestion_clavier);
  glutMainLoop();
  return 0;
}
```

INITIALISATION

```
void init (void)
{
  /* couleur d'effacement */
  glClearColor (0.0, 0.0, 0.0, 0.0);

  /* La matrice de projection... */
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
  /* ou gluOrtho2D (-1.0, 1.0, -1.0, 1.0); ... */
}
```

AFFICHAGE

```
void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glRotatef(angle, 0.0, 1.0, 0.0);
    glBegin(GL_QUADS);
        glColor3f(1.0, 0.0, 0.0);
        glVertex3f(-0.5, -0.5, 0.5); glVertex3f( 0.5, -0.5, 0.5);
        glVertex3f( 0.5,  0.5, 0.5); glVertex3f(-0.5,  0.5, 0.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex3f(-0.5, -0.5, -0.5); glVertex3f( 0.5, -0.5, -0.5);
        glVertex3f( 0.5,  0.5, -0.5); glVertex3f(-0.5,  0.5, -0.5);
    glEnd();
    glutSwapBuffers();
}
```

GESTION DU CLAVIER

```
void gestion_clavier(int touche, int x, int y)
{
  switch (touche)
  {
    case GLUT_KEY_RIGHT:
      angle += 5;
      break;
    case GLUT_KEY_LEFT:
      angle -= 5;
      break;
  }
  /* force le reaffichage */
  glutPostRedisplay();
}
```

OpenGL – Plan

OpenGL – Plan

La définition des points

Les points des primitives sont donnés par les fonctions glVertex∗(). Il en existe beaucoup, suivant le nombre et le type des coordonnés :

- **void** glVertex2f(GLfloat x, GLfloat y); pour un point en 2D avec les coordonnées en flottant.
- **void** glVertex3d(GLdouble x, GLdouble y, GLdouble z); pour un point 3D avec les coordonnées en flottant double précision.
- **void** glVertex4i(GLint x, GLint y, GLint z, GLint w); pour un point 4D avec des coordonnées entières.
- etc.

LES COULEURS

Il est bien entendu possible de spécifier des couleurs pour chaque entité.
Lorsqu'une couleur est donnée, elle reste valable jusqu'au prochain changement de couleur, qui peut avoir lieu entre deux facettes, mais aussi entre deux lignes ou encore entre deux points.
Les fonctions permettant de changer la couleur courante sont :

- **void** glColor3f(GLfloat red, GLfloat green, GLfloat blue); pour une couleur dont les trois composantes rouge, verte et bleue sont un nombre flottant compris entre 0.0 et 1.0.

- ```
  void glColor4ub(GLubyte red, GLubyte green,
                  GLubyte blue, GLubyte alpha);
  ```

  pour une couleur dont les quatre composantes (rouge, verte, bleue, et alpha) sont un nombre entre 0 et 255.

- etc.
  ```

COULEUR - EXEMPLE

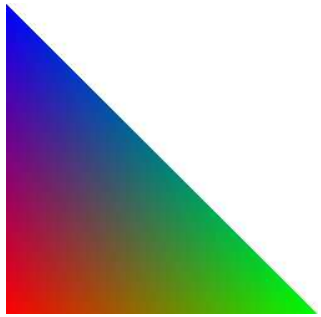
Triangle rouge :

```
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_TRIANGLE);
   glVertex3f(0.0, 0.0, 0.0);
   glVertex3f(1.0, 0.0, 0.0);
   glVertex3f(0.0, 1.0, 0.0);
glEnd();
```

COULEUR - EXEMPLE

Triangle dégradé :

```
glBegin(GL_TRIANGLE);
   glColor3f(1.0, 0.0, 0.0);
   glVertex3f(0.0, 0.0, 0.0);
   glColor3f(0.0, 1.0, 0.0);
   glVertex3f(1.0, 0.0, 0.0);
   glColor3f(0.0, 0.0, 1.0);
   glVertex3f(0.0, 1.0, 0.0);
glEnd();
```



Définition des Normales

La plupart des effets de lumière dépendent de la normale en un point, une face, etc.
Les fonctions suivantes permettent de définir une telle normale :

- glNormal3b();
- glNormal3d();
- glNormal3f();
- glNormal3i();
- glNormal3s();
- glNormal3bv();
- glNormal3dv();
- glNormal3fv();
- glNormal3iv();
- glNormal3sv();

OpenGL – Plan

La déclaration de toutes les primitives se fait de la même façon :

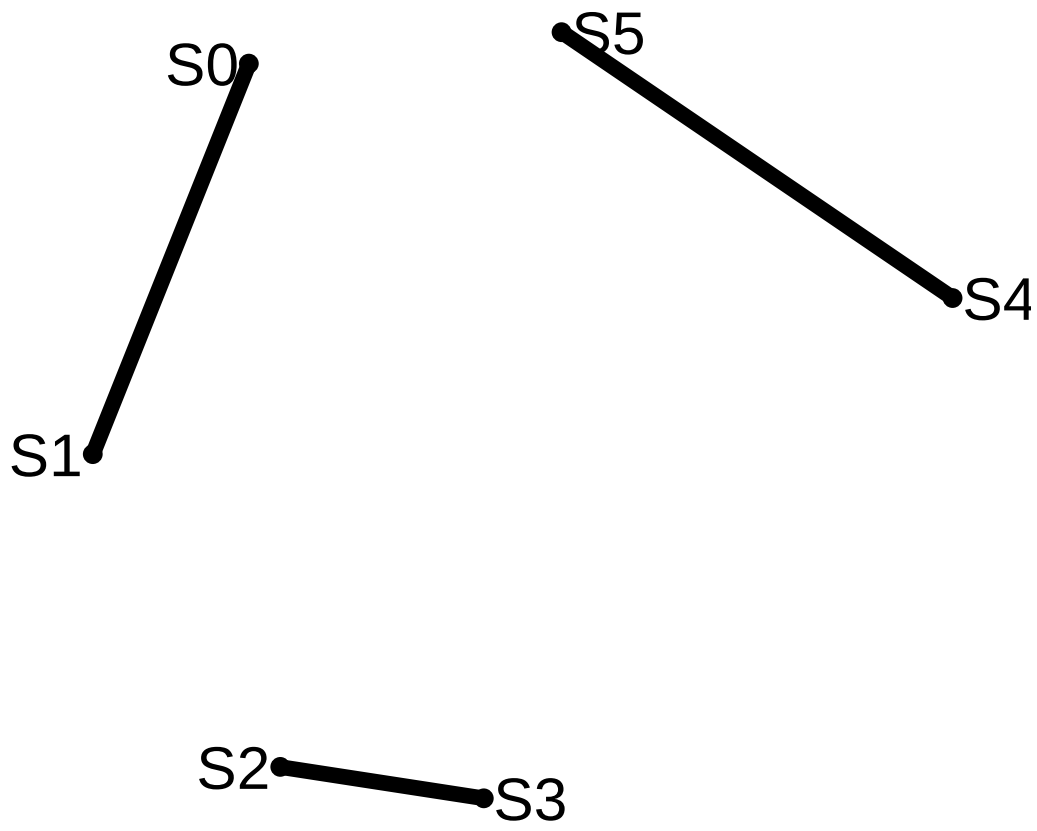
```
glBegin(primitive);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  ...
  glVertex3f(xn, yn, zn);
glEnd();
```

LES POINTS

GL_POINTS

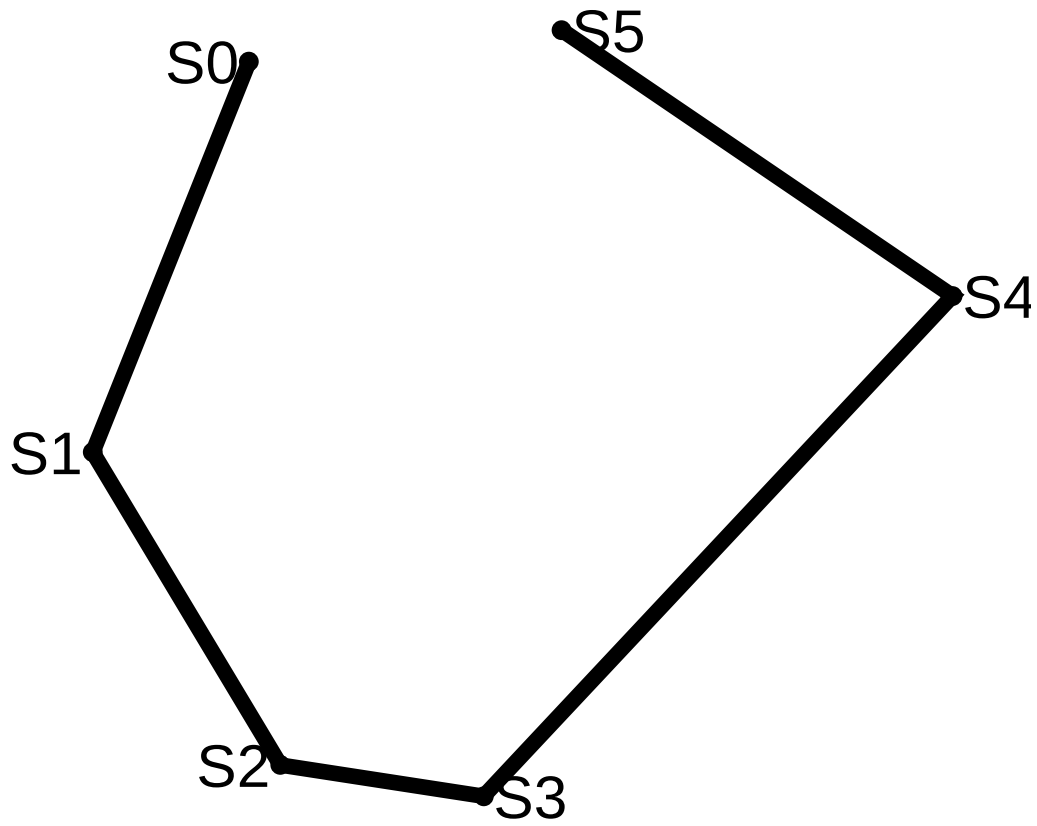
LES LIGNES

GL_LINES



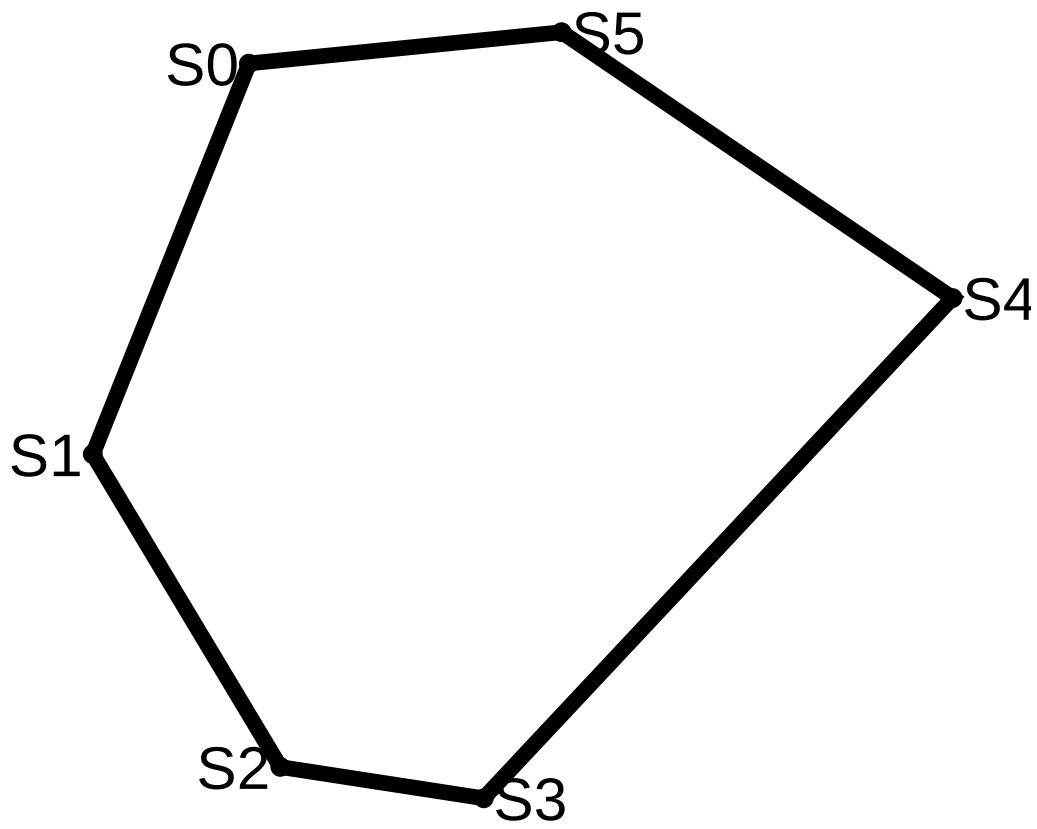
Les chaînes de lignes

GL_LINE_STRIP



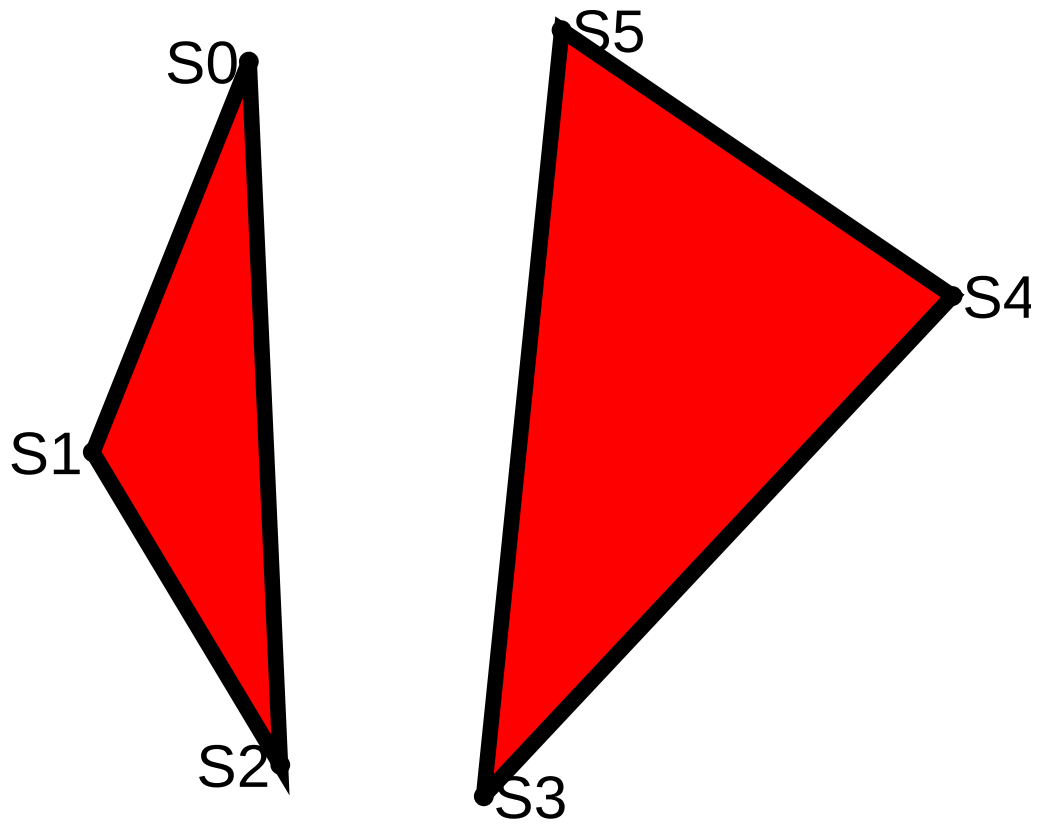
LES BOUCLES DE LIGNES

GL_LINE_LOOP



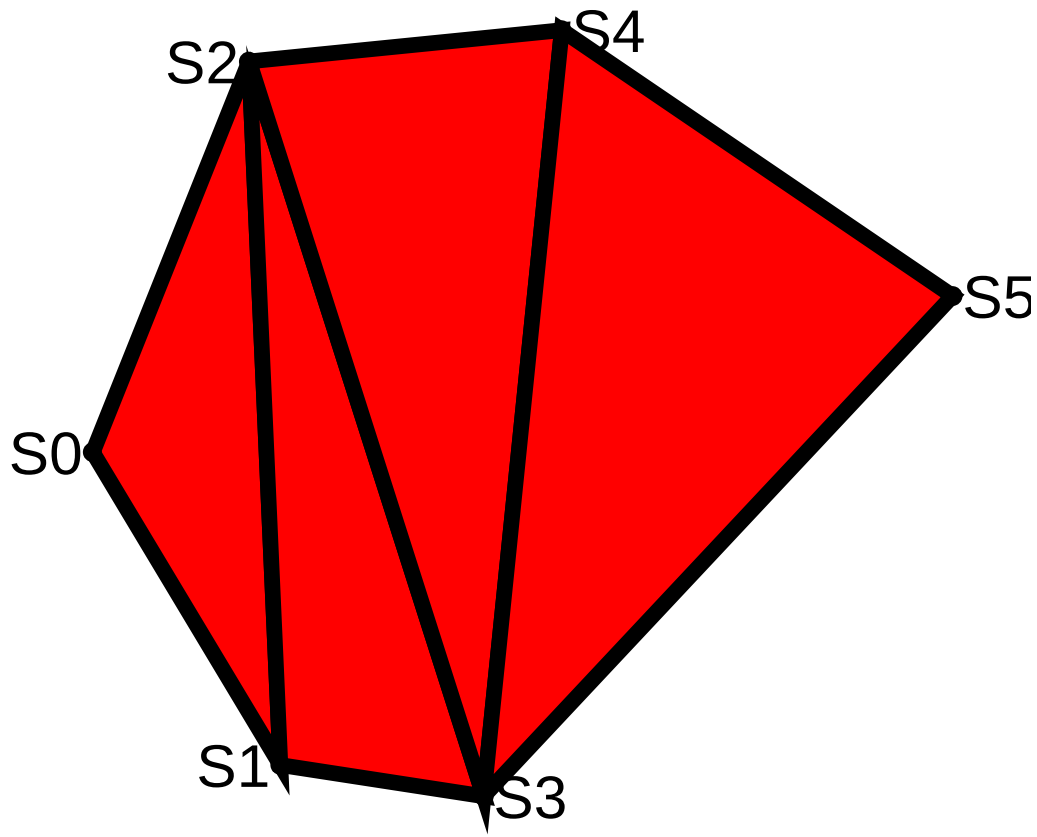
LES TRIANGLES

GL_TRIANGLES



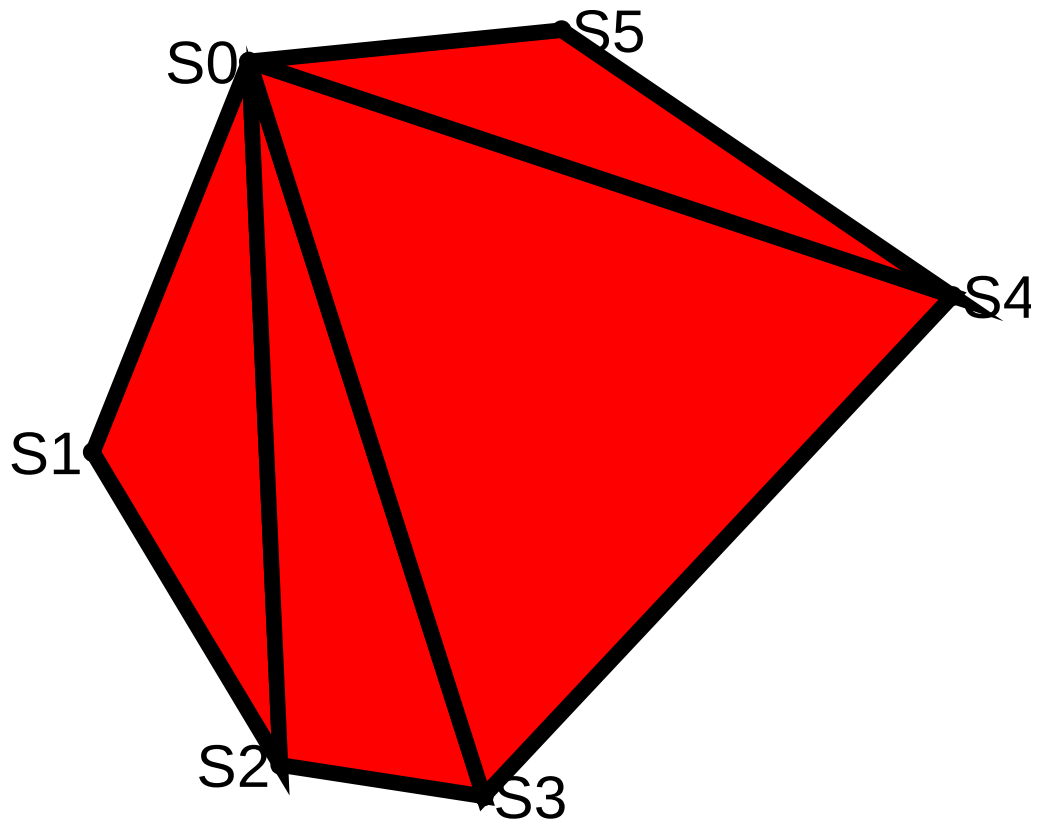
Les chaînes de triangles

GL_TRIANGLE_STRIP



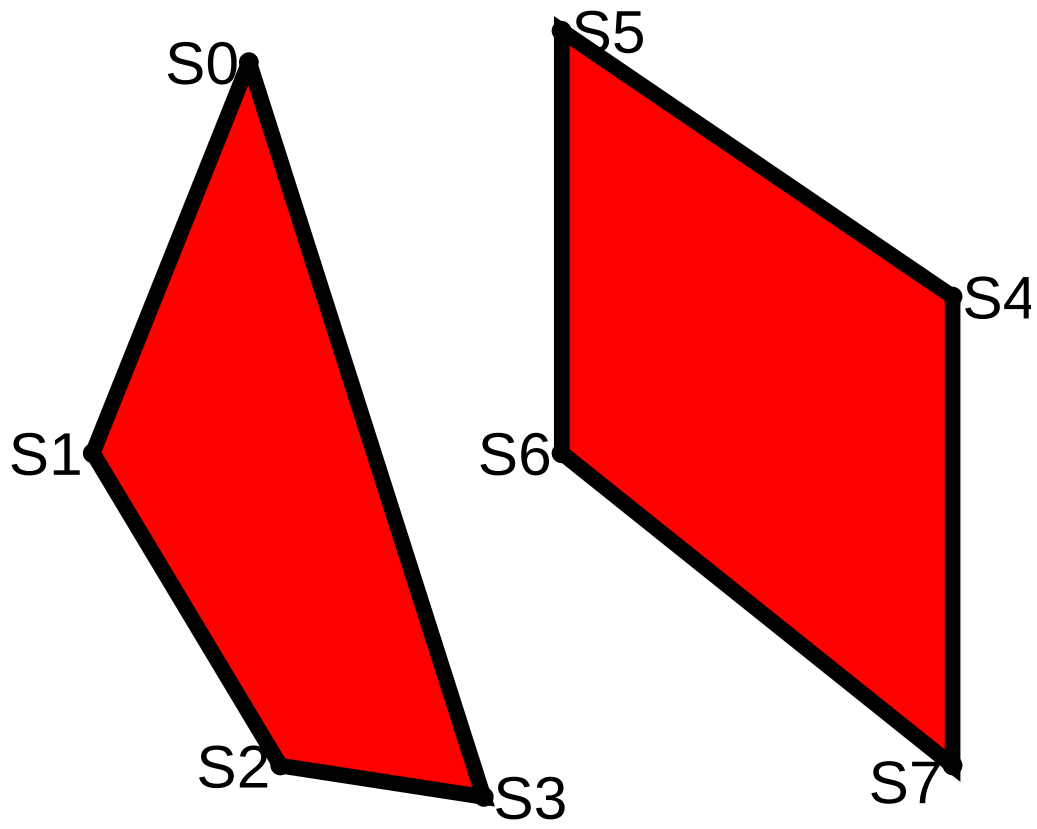
Les éventails de triangles

GL_TRIANGLE_FAN



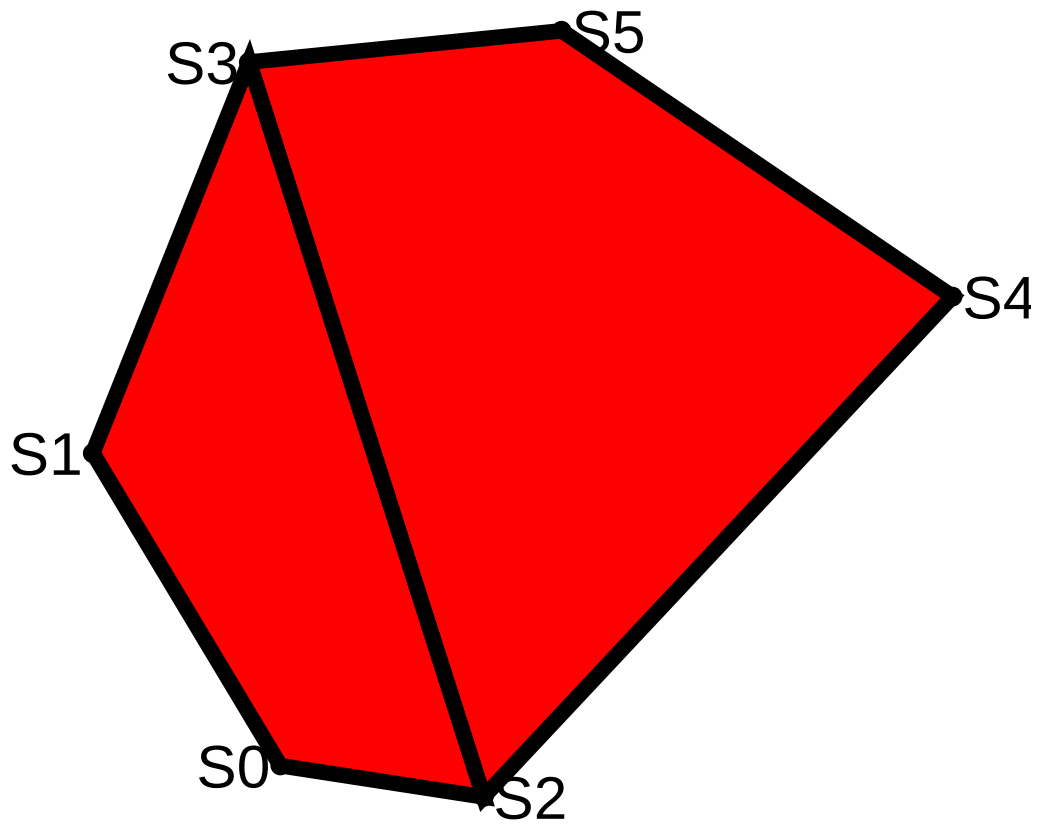
Les quadrilatères

GL_QUADS



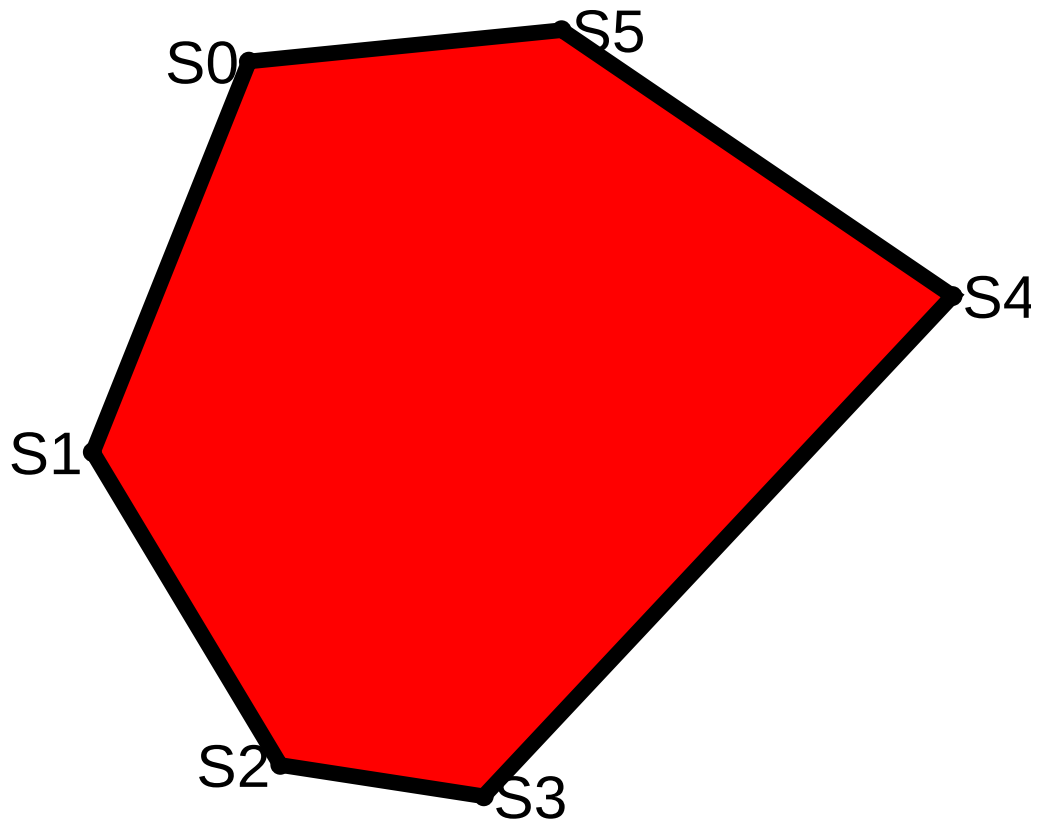
Les chaînes de quadrilatères

GL_QUAD_STRIP



Les polygônes

GL_POLYGON



OpenGL – Plan

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par glMatrixMode(GL_MODELVIEW);

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par `glMatrixMode(GL_MODELVIEW);`
- Reste dans un état.

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par `glMatrixMode(GL_MODELVIEW);`
- Reste dans un état.
- Remise à l'identité par `glLoadIdentity();`

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par `glMatrixMode(GL_MODELVIEW);`
- Reste dans un état.
- Remise à l'identité par `glLoadIdentity();`
- Translation : `glTranslate*();`

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par `glMatrixMode(GL_MODELVIEW);`
- Reste dans un état.
- Remise à l'identité par `glLoadIdentity();`
- Translation : `glTranslate*();`
- Rotation : `glRotate*();`

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par `glMatrixMode(GL_MODELVIEW);`
- Reste dans un état.
- Remise à l'identité par `glLoadIdentity();`
- Translation : `glTranslate*();`
- Rotation : `glRotate*();`
- Affinité : `glScale*();`

La matrice de Modélisation

- Permet de transformer tous les sommets **suivants**.
- Sélection par glMatrixMode(GL_MODELVIEW);
- Reste dans un état.
- Remise à l'identité par glLoadIdentity();
- Translation : glTranslate*();
- Rotation : glRotate*();
- Affinité : glScale*();
- Transformation libre : glMultMatrix*();

EXEMPLES DE TRANSFORMATION

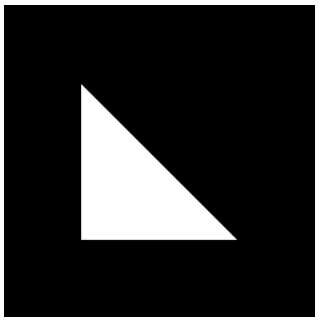
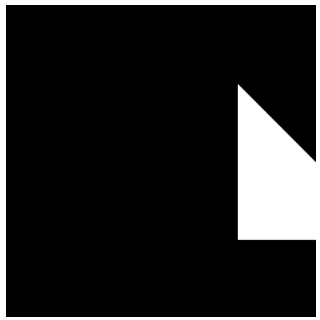
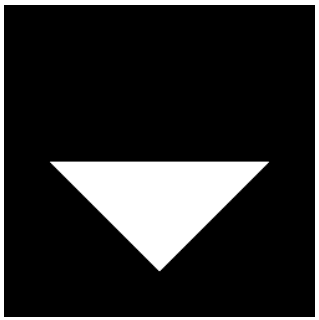


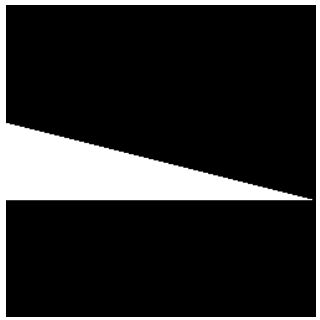
image de base



glTranslatef(1.0, 0.0, 0.0);

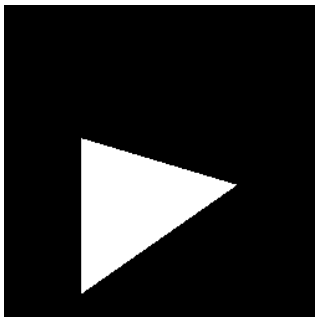


glRotatef(45.0, 0.0, 0.0, 1.0);



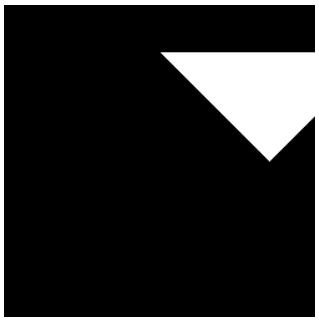
glScalef(2.0, 0.5, 1.0);

TRANSFORMATIONS

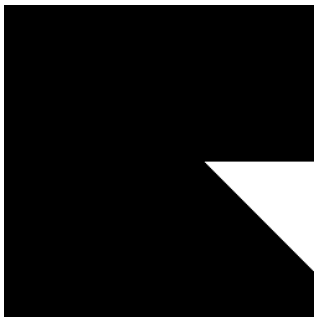


glMultMatrixf(m)

$$m = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0.7 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



glRotatef(45.0, 0.0, 0.0, 1.0);
glTranslatef(1.0, 0.0, 0.0);



glTranslatef(1.0, 0.0, 0.0);
glRotatef(45.0, 0.0, 0.0, 1.0);

MODÉLISATION COMPLEXE

En cas de modélisation complexe, on peut vouloir “dépiler” les dernières transformations de matrices que l’on a effectué.
Par exemple, on modélise un personnage et la modélisation d’un bras demande à ce que l’on commence dans le même repère pour chaque bras.
Pour cela, on peut sauver la matrice courante dans sa pile pour la retrouver par la suite à l’aide des fonctions :

- **void** glPushMatrice (**void**);
- **void** glPopMatrice (**void**);

La matrice de projection

- Décrit la caméra.
- Sélection par glMatrixMode(GL_PROJECTION);
- Reste dans un état.
- Remise à l'identité par glLoadIdentity();
- Mêmes transformations que la matrice de modélisation.
- Plus simple : gluLookAt(ex,ey,ez, cx,cy,cz, upx, upy, upz);

Orthographique :

- glOrtho(left , right , bottom, top , near, far);
- gluOrtho2D(left, right , bottom, top);

La projection

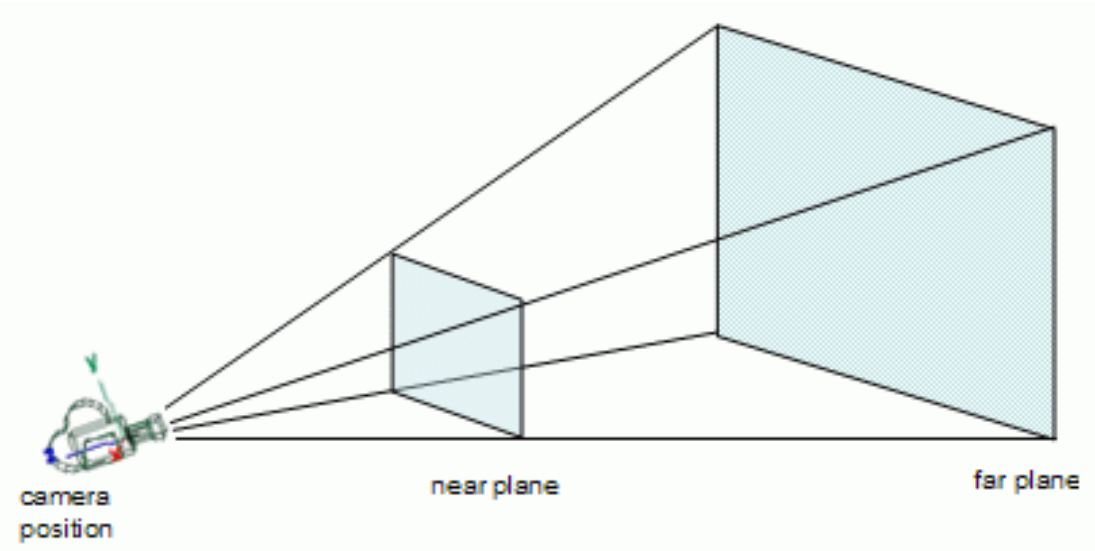
Orthographique :

- glOrtho(left , right , bottom, top , near, far);
- gluOrtho2D(left, right , bottom, top);

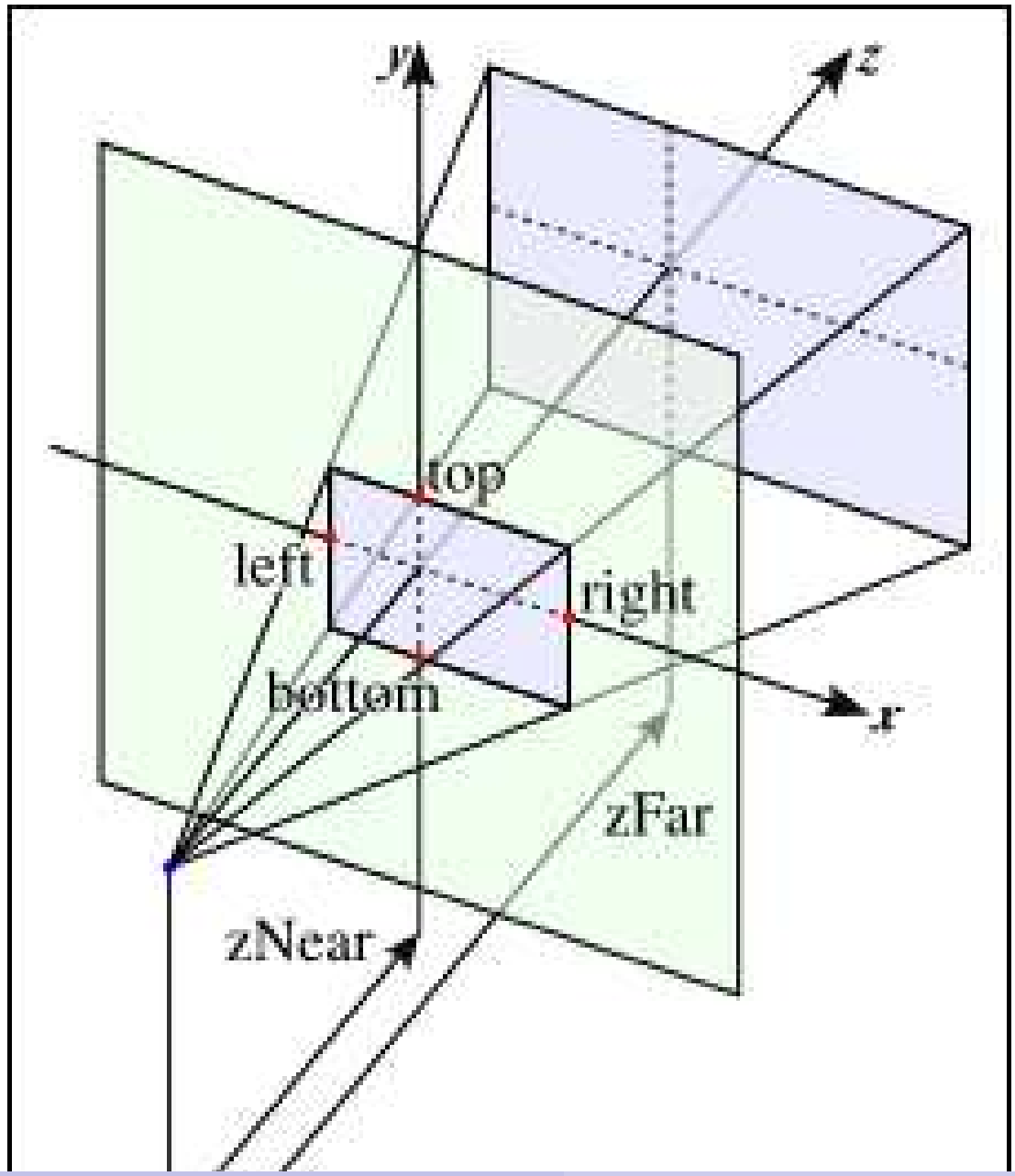
Perspective :

- glFrustum(xmin, xmax, ymin, ymax, zmin, zmax);
- zmin et zmax doivent être strictement positifs.
- tronc de pyramide.
- gluPerspective(ouverture, ratio_XY , zmin, zmax);
- zmin et zmax doivent être strictement positifs.

FRUSTUM



FRUSTUM PARAMÈTRES



OpenGL – Plan

CLIPPING

Les affichages des primitives sont clippés (découpés) suivant le cube d'affichage (donc sur les 6 plans).
Mais on peut aussi ajouter des plans de découpe à l'aide de la fonction :

```
void glClipPlane(GLenum plane,
                 const GLdouble *equation);
```

Utilisé pour des effets de miroir par exemple ou pour accélérer le rendu.

CULLING

Notion très générale permettant de ne pas dessiner ce qui n'est pas visible.
Utilisée dans le cadre d'OpenGL pour choisir quelle(s) face(s) des polygones doivent être dessinées.
Pour utiliser le culling, on appelle glEnable avec la valeur GL_CULL_FACE
Puis on choisit quelle face est automatiquement éliminée (non tracée) avec la fonction

```
void glCullFace(GLenum mode);
```

mode peut prendre les valeurs GL_FRONT ou GL_BACK
Mais la notion de avant (front) et arrière (back) peut être relative, on peut donc choisir quelle face est la face avant avec la fonction :

```
void glFrontFace(GLenum mode);
```

mode peut être soit GL_CW soit GL_CCW (par défaut)

OpenGL – Plan

L'aspect des points et des lignes affichés à l'écran peut être contrôlé par les fonctions suivantes :

```
void glPointSize (GLfloat taille);
void glLineWidth (GLfloat largeur);
void glLineStipple (GLint facteur,
                    GLushort motif);
```

POLYGONE

L'aspect des polygones (à partir du triangle) est contrôlé par les fonctions suivantes :

POLYGONE

L'aspect des polygones (à partir du triangle) est contrôlé par les fonctions suivantes :

- **void** glShadeModel (GLenum mode); permet de choisir si les polygones doivent être unis (si *mode* vaut GL_FLAT) ou en dégradé (si *mode* vaut GL_SMOOTH)

POLYGONE

L'aspect des polygones (à partir du triangle) est contrôlé par les fonctions suivantes :

- **void** glShadeModel (GLenum mode); permet de choisir si les polygones doivent être unis (si *mode* vaut GL_FLAT) ou en dégradé (si *mode* vaut GL_SMOOTH)
- **void** glPolygonStipple (**const** GLubyte *mask); permet de définir un motif de remplissage. Le paramètre est un pointeur vers une zone qui sera interprété comme 32 fois 32 bits. Seuls les bits à 1 seront dessinés.

POLYGONE

L'aspect des polygones (à partir du triangle) est contrôlé par les fonctions suivantes :

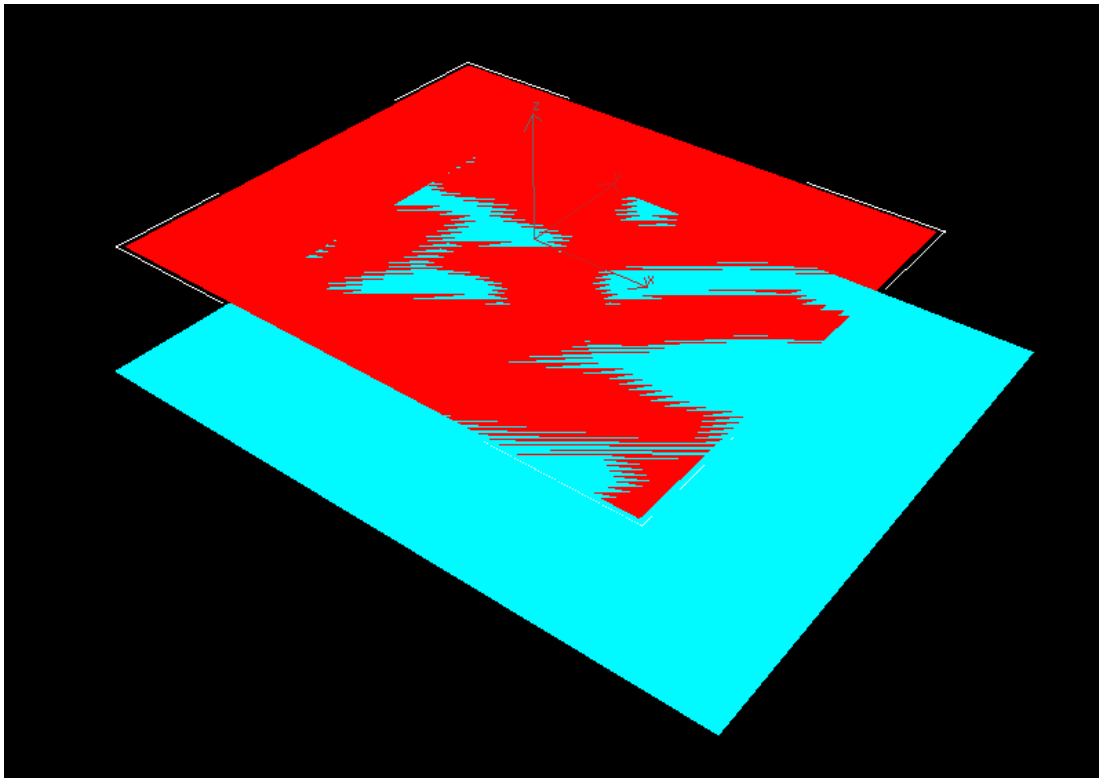
- **void** glShadeModel (GLenum mode); permet de choisir si les polygones doivent être unis (si *mode* vaut GL_FLAT) ou en dégradé (si *mode* vaut GL_SMOOTH)
- **void** glPolygonStipple (**const** GLubyte *mask); permet de définir un motif de remplissage. Le paramètre est un pointeur vers une zone qui sera interprété comme 32 fois 32 bits. Seuls les bits à 1 seront dessinés.
- **void** glPolygonMode (GLenum face, GLenum mode); permet de définir comment seront dessinées les faces. *face* peut prendre les valeurs GL_FRONT, GL_BACK ou GL_FRONT_AND_BACK et *mode* peut prendre les valeurs GL_POINT, GL_LINE, ou GL_FILL

POLYGONE

L'aspect des polygones (à partir du triangle) est contrôlé par les fonctions suivantes :

- **void** glShadeModel (GLenum mode); permet de choisir si les polygones doivent être unis (si *mode* vaut GL_FLAT) ou en dégradé (si *mode* vaut GL_SMOOTH)
- **void** glPolygonStipple (**const** GLubyte *mask); permet de définir un motif de remplissage. Le paramètre est un pointeur vers une zone qui sera interprété comme 32 fois 32 bits. Seuls les bits à 1 seront dessinés.
- **void** glPolygonMode (GLenum face, GLenum mode); permet de définir comment seront dessinées les faces. *face* peut prendre les valeurs GL_FRONT, GL_BACK ou GL_FRONT_AND_BACK et *mode* peut prendre les valeurs GL_POINT, GL_LINE, ou GL_FILL
- **void** glPolygonOffset (GLfloat factor, GLfloat units); permet de lutter contre le Z-fighting.

Z FIGHTING



OpenGL – Plan

Sources de lumières

- Désactivées par défaut.

SOURCES DE LUMIÈRES

- Désactivées par défaut.
- Activation globale par `glEnable(GL_LIGHTING);`

SOURCES DE LUMIÈRES

- Désactivées par défaut.
- Activation globale par glEnable(GL_LIGHTING);
- Une lumière ambiante et 8 sources positionnelles ou directionnelles.

SOURCES DE LUMIÈRES

- Désactivées par défaut.
- Activation globale par `glEnable(GL_LIGHTING);`
- Une lumière ambiante et 8 sources positionnelles ou directionnelles.
- Lumère ambiante : `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, couleur)`

SOURCES DE LUMIÈRES

- Désactivées par défaut.
- Activation globale par `glEnable(GL_LIGHTING);`
- Une lumière ambiante et 8 sources positionnelles ou directionnelles.
- Lumère ambiante : `glLightModelfv(GL_LIGHT_MODEL_AMBIENT, couleur)`
- Activation d'une source de lumière supplémentaire : `glEnable(GL_LIGHT0);` à `glEnable(GL_LIGHT7);`.

SOURCES DE LUMIÈRES

- Désactivées par défaut.
- Activation globale par glEnable(GL_LIGHTING);
- Une lumière ambiante et 8 sources positionnelles ou directionnelles.
- Lumère ambiante : glLightModelfv(GL_LIGHT_MODEL_AMBIENT, couleur)
- Activation d'une source de lumière supplémentaire : glEnable(GL_LIGHT0); à glEnable(GL_LIGHT7);.
- Définition des paramètres d'une source de lumière à l'aide de glLightfv ();

PARAMÈTRES D'UNE SOURCE

Syntaxe : `glLightfv(numero, parametre, valeurs);`

- numero : numéro de la source de `GL_LIGHT0` à `GL_LIGHT7`.
- valeurs : un tableau de nombres réels.

Et “parametre” peut prendre l'une des valeurs suivantes :

- `GL_POSITION`

  valeurs est alors un tableau de quatre coordonnées donnant la position de la source, dans le système du modèle. La dernière coordonnée nulle correspond à une lumière directionnelle. Valeur par défaut : (0, 0, 1, 0).

PARAMÈTRES D'UNE SOURCE

Syntaxe : glLightfv(numero, parametre, valeurs);

- numero : numéro de la source de GL_LIGHT0 à GL_LIGHT7.
- valeurs : un tableau de nombres réels.

Et "parametre" peut prendre l'une des valeurs suivantes :

- GL_POSITION

  valeurs est alors un tableau de quatre coordonnées donnant la position de la source, dans le système du modèle. La dernière coordonnée nulle correspond à une lumière directionnelle.
  Valeur par défaut : (0, 0, 1, 0).

- GL_AMBIENT

  valeurs est un tableau de quatre composantes de couleurs (RGBA) donnant la valeur ambiante de cette lumière.
  Valeur par défaut : (0, 0, 0, 1) noir opaque (aucun effet)

PARAMÈTRES D'UNE SOURCE

- GL_DIFFUSE

  valeur est un tableau de quatre composantes (RGBA) donnant la quantité de lumière diffuse émise par cette lumière.
  Valeur par défaut : (1, 1, 1, 1) pour GL_LIGHT0 et (0, 0, 0, 0) pour les autres.

- GL_SPECULAR

  Comme précédemment, mais pour la composante spéculaire.

Les lumières spot

Une source de lumière peut être un spot lumineux, comme une lampe torche.

- GL_SPOT_DIRECTION
  valeur est alors un tableau de trois coordonnées donnant la direction principale du faisceau lumineux.
  Valeur par défaut : (0, 0, -1).

Les lumières spot

Une source de lumière peut être un spot lumineux, comme une lampe torche.

- GL_SPOT_DIRECTION
  valeur est alors un tableau de trois coordonnées donnant la direction principale du faisceau lumineux.
  Valeur par défaut : (0, 0, -1).
- GL_SPOT_EXPONENT
  valeur est alors un seul nombre indiquant la distribution du faisceau lumineux. Une valeur forte indique un faisceau plus fin.
  La valeur par défaut est 1.

Les lumières spot

Une source de lumière peut être un spot lumineux, comme une lampe torche.

- GL_SPOT_DIRECTION
  valeur est alors un tableau de trois coordonnées donnant la direction principale du faisceau lumineux.
  Valeur par défaut : (0, 0, -1).
- GL_SPOT_EXPONENT
  valeur est alors un seul nombre indiquant la distribution du faisceau lumineux. Une valeur forte indique un faisceau plus fin.
  La valeur par défaut est 1.
- GL_SPOT_CUTOFF
  valeur est alors interprété comme un angle (entre 0 et 90 degrés) qui est l'angle d'ouverture du faisceau lumineux.
  La valeur par défaut est la valeur très particulière 180 degrés.

L'atténuation lumineuse

- GL_CONSTANT_ATTENUATION (kc)
- GL_LINEAR_ATTENUATION (kl)
- GL_QUADRATIC_ATTENUATION (kq)

Ces trois paramètres permettent de définir une atténuation lumineuse d'une source. L'atténuation sera alors calculée suivant la formule :

$$f = \frac{1}{kc + kl \cdot d + kq \cdot d^2}$$

Les matériaux

Les primitives OpenGL (triangles, quadrilatères, points, etc.) peuvent avoir un comportement particulier avec la lumière.
On a déjà vu la possibilité de donner une couleur avec `glColor3f()`.
Pour les autres aspects, on utilisera :

```
glMaterialfv(face, parametre, valeurs);
```

Qui fonctionne un peu comme `glLightfv()`, pour chaque objet.
Le premier argument, face, indique quelle(s) face est impliquée par la fonction :

- `GL_FRONT` pour la face visible,
- `GL_BACK` pour la face cachée,
- `GL_FRONT_AND_BACK` pour les deux.

Les paramètres de matériaux

Le paramètre central définit quel aspect du matériau est affecté. Il peut valoir :

- GL_AMBIENT
  a la même signification que pour les lumières. La véritable valeur ambiante est le produit de ce paramètre par la somme des valeurs ambiantes des lumières qui éclairent l'objet.
  La valeur par défaut est : (0.2, 0.2, 0.2, 1.0).

Les paramètres de matériaux

Le paramètre central définit quel aspect du matériau est affecté. Il peut valoir :

- GL_AMBIENT
  a la même signification que pour les lumières. La véritable valeur ambiante est le produit de ce paramètre par la somme des valeurs ambiantes des lumières qui éclairent l'objet.
  La valeur par défaut est : (0.2, 0.2, 0.2, 1.0).
- GL_DIFFUSE
  Même remarque, mais pour la partie diffuse.
  La valeur par défaut est : (0.8, 0.8, 0.8, 1.0).

Les paramètres de matériaux

Le paramètre central définit quel aspect du matériau est affecté. Il peut valoir :

- GL_AMBIENT
  a la même signification que pour les lumières. La véritable valeur ambiante est le produit de ce paramètre par la somme des valeurs ambiantes des lumières qui éclairent l'objet.
  La valeur par défaut est : (0.2, 0.2, 0.2, 1.0).
- GL_DIFFUSE
  Même remarque, mais pour la partie diffuse.
  La valeur par défaut est : (0.8, 0.8, 0.8, 1.0).
- GL_SPECULAR
  Idem, pour la partie spéculaire.
  La valeur par défaut est : (0, 0, 0, 1).

Les paramètres de matériaux

- GL_EMISSION

  Valeur est un tableau de 4 composantes (RGBA) indiquant quelle quantité de lumière est émise par l'objet.
  La valeur par défaut est : (0, 0, 0, 1).

Les paramètres de matériaux

- GL_EMISSION

  Valeur est un tableau de 4 composantes (RGBA) indiquant quelle quantité de lumière est émise par l'objet.
  La valeur par défaut est : (0, 0, 0, 1).

- GL_SHININESS

  Valeur est une valeur entre 0 et 128, définissant la brilliance de l'objet. La valeur par défaut est 0.

Les paramètres de matériaux

- GL_EMISSION

  Valeur est un tableau de 4 composantes (RGBA) indiquant quelle quantité de lumière est émise par l'objet.
  La valeur par défaut est : (0, 0, 0, 1).

- GL_SHININESS

  Valeur est une valeur entre 0 et 128, définissant la brilliance de l'objet. La valeur par défaut est 0.

- GL_AMBIENT_AND_DIFFUSE

  permet de définir à la fois la partie ambiante et la partie diffuse.

OpenGL – Plan

BLENDING

L'affichage d'une primitive à l'écran se fait normalement en remplaçant ce qu'il y avait déjà (oublions le Z-Buffer pour l'instant). A l'aide du blending, il est possible de combiner ce qu'il y a déjà à l'écran et ce que l'on souhaite y ajouter.
Cela ce met en place avec un glEnable(GL_BLEND); et en choisissant la fonction de blending pour la source (ce que l'on dessine) et la destination (l'écran).
La couleur résultante sera calculée à l'aide de la formule suivante :

$$C = C_s \cdot f_s + C_d \cdot f_d$$

Où C_s est la couleur de la primitive en un point de l'écran, C_d celle de l'écran en ce même point, et f_s et f_d sont les paramètres de la fonction suivante :

- **void** glBlendFunc(GLenum fs, GLenum fd);

f_s et f_d peuvent prendre les valeurs suivantes :

GL_ZERO	$(0, 0, 0)$
GL_ONE	$(1, 1, 1)$
GL_SRC_COLOR	(R_s, V_s, B_s)
GL_ONE_MINUS_SRC_COLOR	$(1, 1, 1) - (R_s, V_s, B_s)$
GL_DST_COLOR	(R_d, V_d, B_d)
GL_ONE_MINUS_DST_COLOR	$(1, 1, 1) - (R_d, V_d, B_d)$
GL_SRC_ALPHA	(A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPHA	$(1, 1, 1) - (A_s, A_s, A_s)$
GL_DST_ALPHA	(A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPHA	$(1, 1, 1) - (A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	$(i, i, i), i = \min(A_s, 1 - A_d)$
GL_CONSTANT_COLOR	(R_c, V_c, B_c)
GL_ONE_MINUS_CONSTANT_COLOR	$(1, 1, 1) - (R_c, V_c, B_c)$
GL_CONSTANT_ALPHA	(A_c, A_c, A_c)
GL_ONE_MINUS_CONSTANT_ALPHA	$(1, 1, 1) - (A_c, A_c, A_c, A_c)$