

Opérations sur les fragments

Opérations sur les fragments

Les fragments représentent un état graphique intermédiaire entre la sortie du pipeline graphique et les pixels à l'écran :

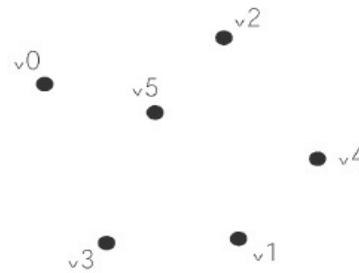
- Une *primitive* est composée de sommets
- Un *fragment* est un élément de géométrie de la taille du pixel
- Un *pixel* (« *picture element* ») est une couleur (+profondeur) à l'écran

primitives ---> fragments ---> pixels

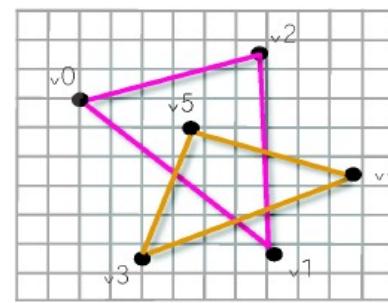
Le tramage des primitives
crée des fragments

Les fragments qui satisfont
certains tests deviennent des pixels

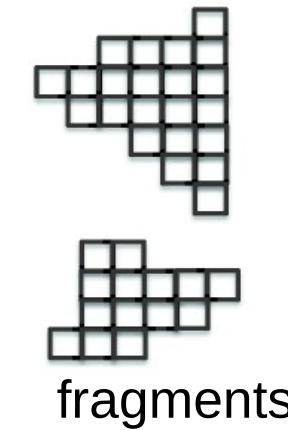
Opérations sur les fragments



sommets



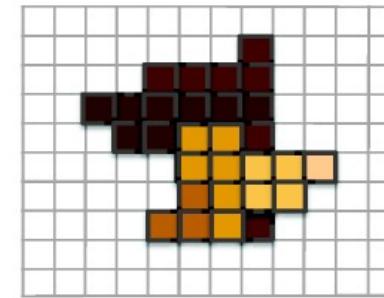
primitive



fragments



fragments (+couleur)



pixels

Opérations sur les fragments

Un fragment doit passer avec succès plusieurs étapes avant de (possiblement) devenir un pixel:

- génération d'un coordonnée de texture
- calcul du brouillard (*fog*)
- antirénelage (*antialiasing*)
- test de découpage (*scissor*)
- test du alpha
- test du stencil
- test de profondeur (*depth buffer*)
- fusion de couleurs (*blending*)
- opérations logiques
- application du masque de couleur

Chaque opération peut être individuellement activée ou désactivée

Opérations sur les fragments

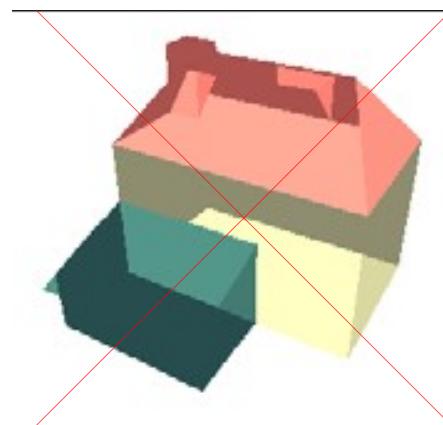
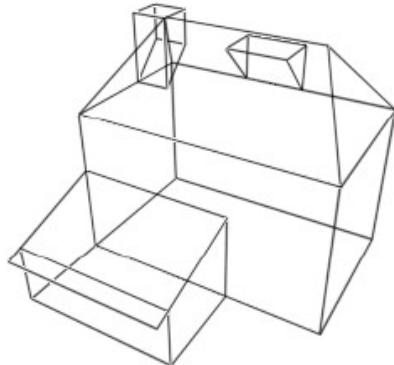
Parmi ces étapes, les tests à réussir pour que le fragment devienne pixel sont:

- test de découpage (*scissor*): élimine (*discard*) les fragments en dehors d'un rectangle aligné avec la fenêtre spécifiée par le programmeur.
- test du alpha: basé sur la valeur de la composante A de la couleur RGBA du fragment.
- test du stencil: le programmeur spécifie une valeur de référence et le test compare la valeur dans le stencil et la référence.
- test de profondeur (*depth buffer*): test basé sur la valeur de la profondeur du fragment. Le programmeur décide du test de comparaison avec le tampon de profondeur.

Suppression des parties cachées (test de profondeur)

Suppression des parties cachées

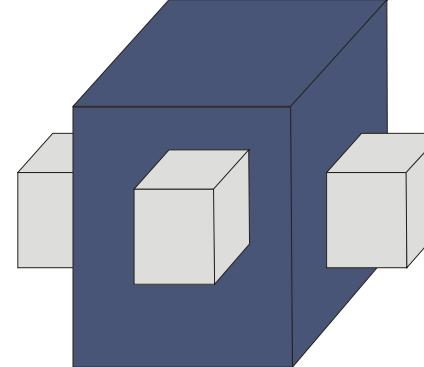
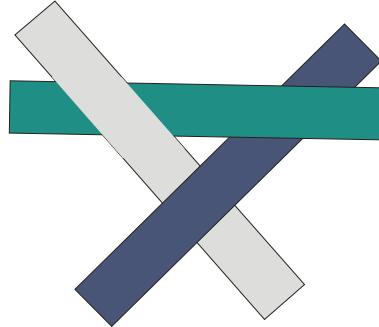
Problème : on veut éviter le résultat du centre



- *Solution 1 : l'algorithme du peintre*
- *Solution 2 : le test de profondeur*

Suppression des parties cachées : l'algorithme du peintre

- ➊ Dessiner les objets du plus éloigné au plus proche
 - solution très utilisée en 2D (la distance est facile à calculer !)
 - plus difficile en 3D
 - quelques cas pathologiques



Suppression des parties cachées : le test de profondeur

Utilisation d'un deuxième tampon (en plus du tampon de couleurs) :
le tampon de profondeur (*depth buffer, Z buffer*)

- Vérifier la profondeur du fragment avant de modifier le pixel

```
if ( fragment.z < tamponProfondeur(i,j).z )
{
    tamponProfondeur(i,j).z = fragment.z;
    tamponCouleur(i,j).color = fragment.color;
}
```

- Les valeurs sont comprises entre [0,1] (0 = plan avant, 1 = plan arrière)
- Le tampon de profondeur doit être créé lors de la création de la fenêtre et il doit généralement être effacé à chaque affichage

Utilisation du tampon de profondeur (OpenGL)

```
void display()
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    // dessiner
    ...
    glutSwapBuffers();
}

void main( int argc, char **argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DEPTH | GLUT_DOUBLE );
    glutCreateWindow( argv[0] );
    glClearColor( 0.0, 0.0, 0.0, 1.0 );
    glEnable( GL_DEPTH_TEST );
    glutDisplayFunc( display );
    glutMainLoop();
}
```

Utilisation du tampon de profondeur (OpenGL)

Le choix du test de profondeur :

```
void glDepthFunc( GLenum func );
```

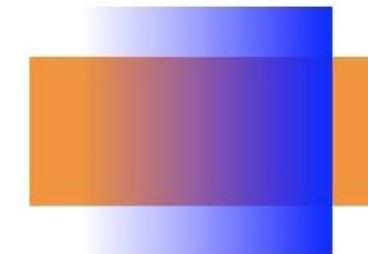
où *func* spécifie la fonction à utiliser pour la comparaison

- *func* peut être `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL` ou `GL_ALWAYS`
- `GL_LESS` est la fonction par défaut

Fusion de couleurs

Fusion de couleurs opacité / transparence -- RGBA

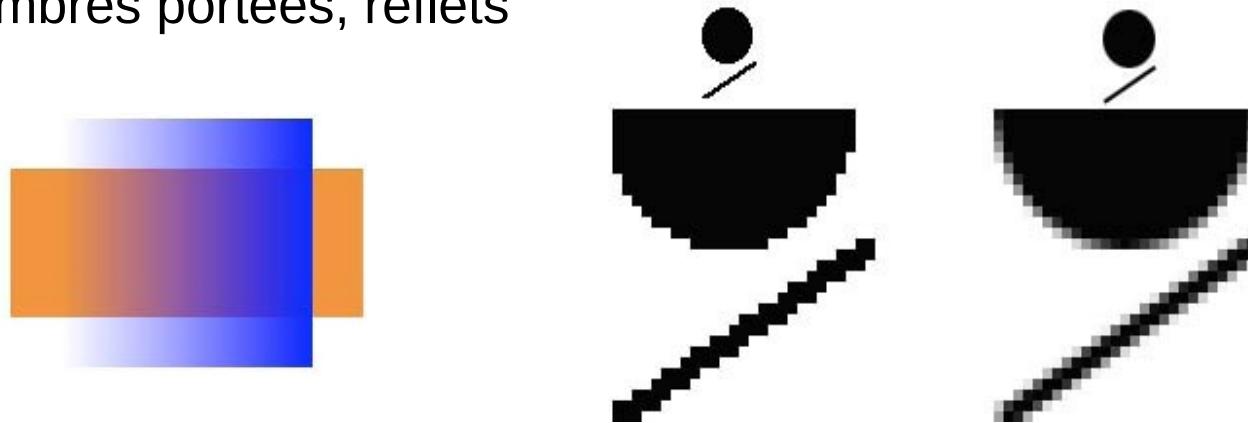
- La composante alpha de RGBA mesure l'opacité du fragment
 - 0.0 = transparent
 - 1.0 = opaque
 - (Les valeurs intermédiaires sont permises.)
- C'est la valeur alpha dans la spécification des couleurs RGBA



```
glColor4{b s f l d ub us ui} ( TYPE r,  
                                TYPE g,  
                                TYPE b,  
                                TYPE alpha );
```

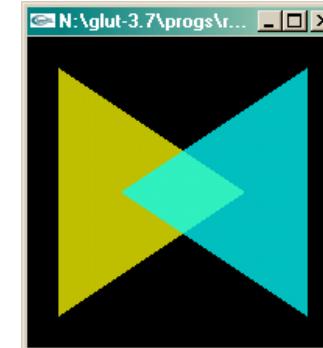
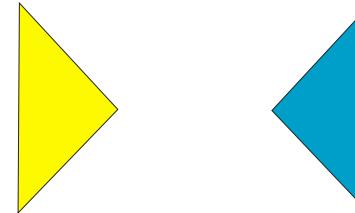
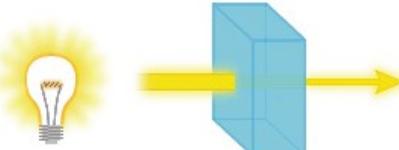
Fusion de couleurs

- But : fusionner des couleurs pour obtenir des effets comme la transparence des objets
- A quoi sert la composante alpha ?
 - simuler des objets transparents ou translucides
 - composer des images
 - faire de l'anticrénelage (*antialiasing*)
 - ombres portées, reflets



Fusion de couleurs

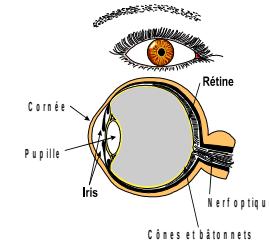
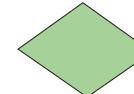
- Surfaces transparentes ou translucides : opacité moindre, i.e. α petit ($\alpha \rightarrow 0$)
 - Exemple : vitre verte laissant passer 80% de la lumière
 $\alpha = 20\%$
 - Objet vu : combinaison de 20% de la vitre et 80% de l'objet derrière
 - Fusion : combinaison des valeurs chromatiques de la source et de la destination





Fusion de couleurs

- Action : spécifier une fonction de fusion en combinant des valeurs chromatiques de source et de destination.
- Choisir une fonction pour mélanger *source* et *destination*
 - La *source* : les fragments qui arrivent du tramage
 - La *destination* : les pixels déjà affichés
 - Résultat (de façon générale) :
 - facteur fusion de la *source* : (S_r, S_v, S_b, S_a)
 - facteur fusion de la *destination* : (D_r, D_v, D_b, D_a)
 - résultat : rouge = $R_s S_r + R_d D_r$,
vert = $V_s S_v + V_d D_v$,
bleu = $B_s S_b + B_d D_b$,
 - alpha = $\alpha_s S_\alpha + \alpha_d D_\alpha$



Fusion de couleurs

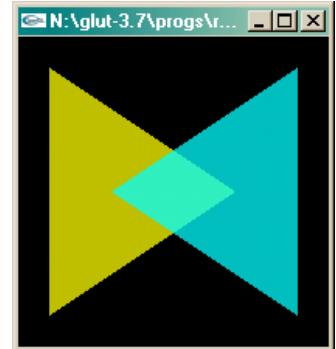
La fonction de mélange :

```
glBlendFunc( GLenum sfactor, GLenum dfactor );
```

sfactor : facteur fusion de source

dfactor : facteur fusion de destination

GL_ONE et GL_ZERO sont les valeurs de défaut



Fusion de couleurs

- ♦ Facteurs de fusion possibles:

GL_ZERO	(0, 0, 0, 0)
GL_ONE	(1, 1, 1, 1)
GL_SRC_COLOR	(Rs, Gs, Bs, As)
GL_ONE_MINUS_SRC_COLOR	(1, 1, 1, 1) - (Rs, Gs, Bs, As)
GL_DST_COLOR	(Rd, Gd, Bd, Ad)
GL_ONE_MINUS_DST_COLOR	(1, 1, 1, 1) - (Rd, Gd, Bd, Ad)
GL_SRC_ALPHA	(As, As, As, As)
GL_ONE_MINUS_SRC_ALPHA	(1, 1, 1, 1) - (As, As, As, As)
GL_DST_ALPHA	(Ad, Ad, Ad, Ad)
GL_ONE_MINUS_DST_ALPHA	(1, 1, 1, 1) - (Ad, Ad, Ad, Ad)
GL_CONSTANT_COLOR	(Rc, Gc, Bc, Ac)
GL_ONE_MINUS_CONSTANT_COLOR	(1, 1, 1, 1) - (Rc, Gc, Bc, Ac)
GL_CONSTANT_ALPHA	(Ac, Ac, Ac, Ac)
GL_ONE_MINUS_CONSTANT_ALPHA	(1, 1, 1, 1) - (Ac, Ac, Ac, Ac)
GL_SRC_ALPHA_SATURATE	(i, i, i, 1) où i = min (As, kA - Ad)

Fusion de couleurs

La variable d'état booléenne `GL_BLEND` sert à activer cette fonctionnalité :

- Activer :

```
glEnable( GL_BLEND );
```

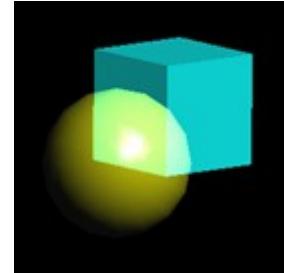
- Désactiver :

```
glDisable( GL_BLEND );
```

`alpha` : aucun effet si la fusion (*blending*) n'est pas activée



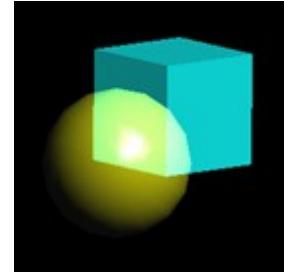
Fusion de couleurs



Usage du tampon de profondeur:

- Le tampon de profondeur stocke la distance entre le point de vue et la portion de l'objet occupant un pixel donné. Lorsqu'un autre objet doit s'ajouter au même pixel, il ne sera dessiné que s'il est plus proche du point de vue, et dans ce cas, c'est sa distance qui sera stockée dans le tampon de profondeur
- Si une même scène contient à la fois des objets opaques et des objets translucides, il y a un *problème* si on laisse les objets translucides masquer des objets opaques ...
- Il faut donc d'abord dessiner les *objets opaques*, en activant le tampon de profondeur en lecture / écriture avec `glDepthMask(GL_TRUE)` (qui correspond au fonctionnement par défaut) pour le mettre à jour au fur et à mesure de l'ajout des objets
- Il faut ensuite activer le tampon de profondeur en lecture seule (par `glDepthMask(GL_FALSE)`) pour ajouter les *objets transparents* pour ne pas modifier les valeurs de z stockées

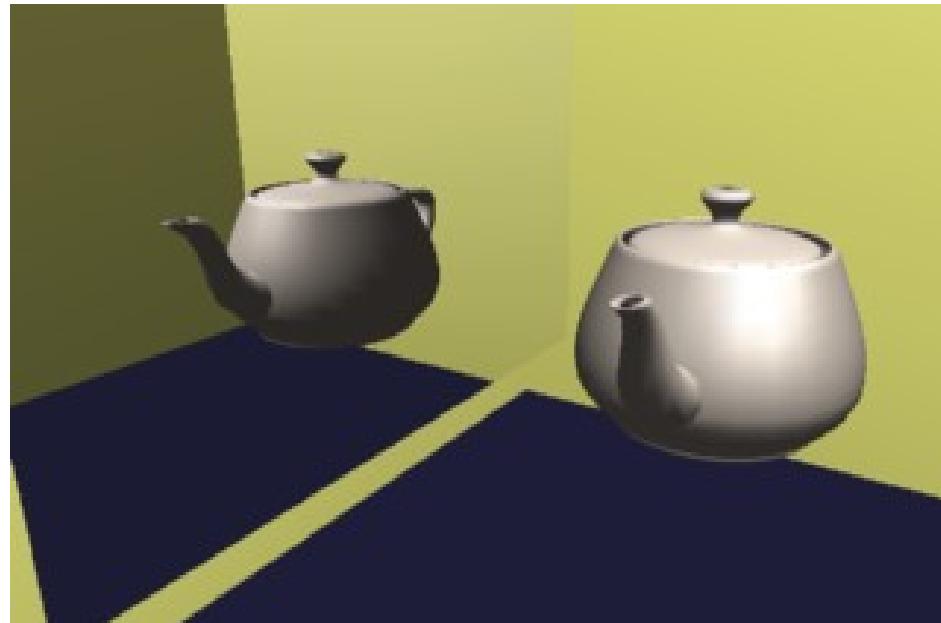
Fusion de couleurs



```
void display(void)
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    // TRACER LA SPHÈRE (opaque et en arrière)
    glPushMatrix();
        glTranslatef( -0.15, -0.15, solidZ );
        TracerSphere( );
    glPopMatrix();
    // TRACER LE CUBE (transparent et en avant de la sphère)
    glPushMatrix();
        glTranslatef( 0.15, 0.15, transparentZ );
        glRotatef( 15.0, 1.0, 1.0, 0.0 );
        glRotatef( 30.0, 0.0, 1.0, 0.0 );
        glEnable( GL_BLEND );
        glDepthMask( GL_FALSE );
        glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
        TracerCube( );
        glDepthMask( GL_TRUE );
        glDisable( GL_BLEND );
    glPopMatrix();
}
```

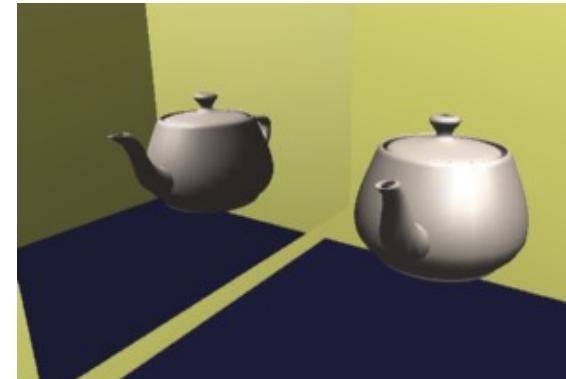
Fusion de couleurs effet de miroir

- La réflexion d'un objet est le même objet mais inversé
- Pour créer un effet de miroir, il faudra donc tracer l'objet deux fois, mais en appliquant une symétrie par rapport à un plan



Fusion de couleurs effet de miroir

- On affiche (fusion de couleurs) l'image inversée avec un rectangle *transparent* représentant le miroir dans la scène originale

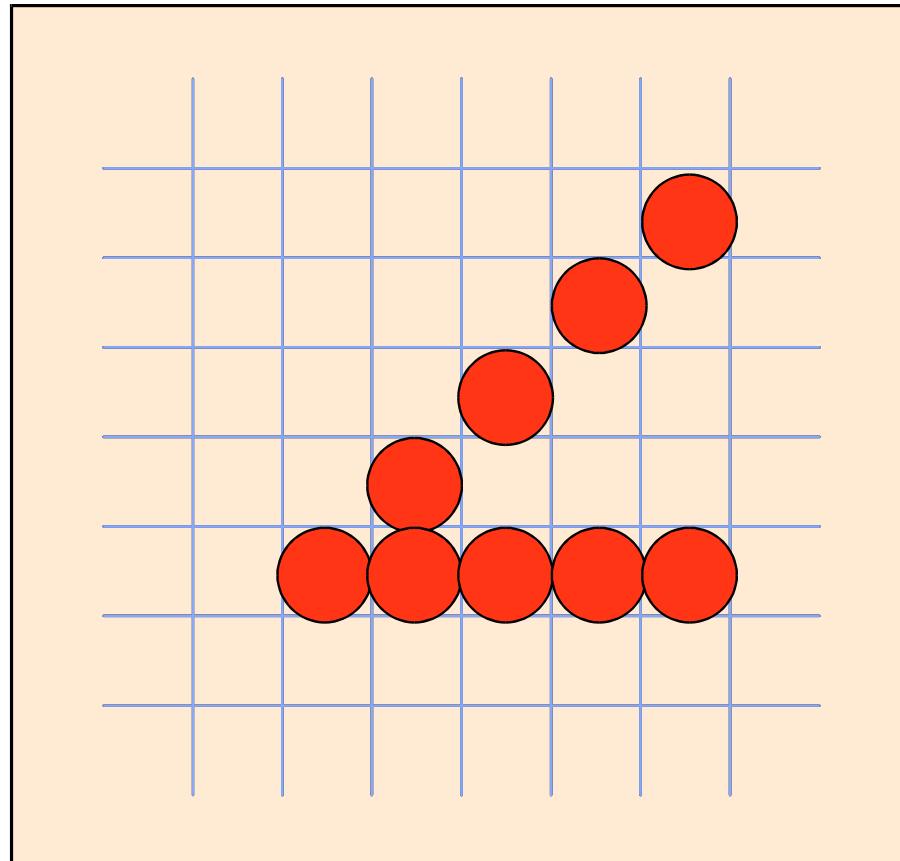


- Quelques soucis:
 - L'inversion est faite avec `glScale(1.0, -1.0, 1.0)`;
 - Les polygones sont tracés à l'envers et les faces avant sont inversées
 - Il ne faut pas que le dessin inversé déborde le miroir
 - On limitera soi-même le dessin ou on utilisera le stencil (dans les prochaines pages)
 - (L'éclairage devra aussi être inversé.)

Anticrénelage (*antialiasing*)

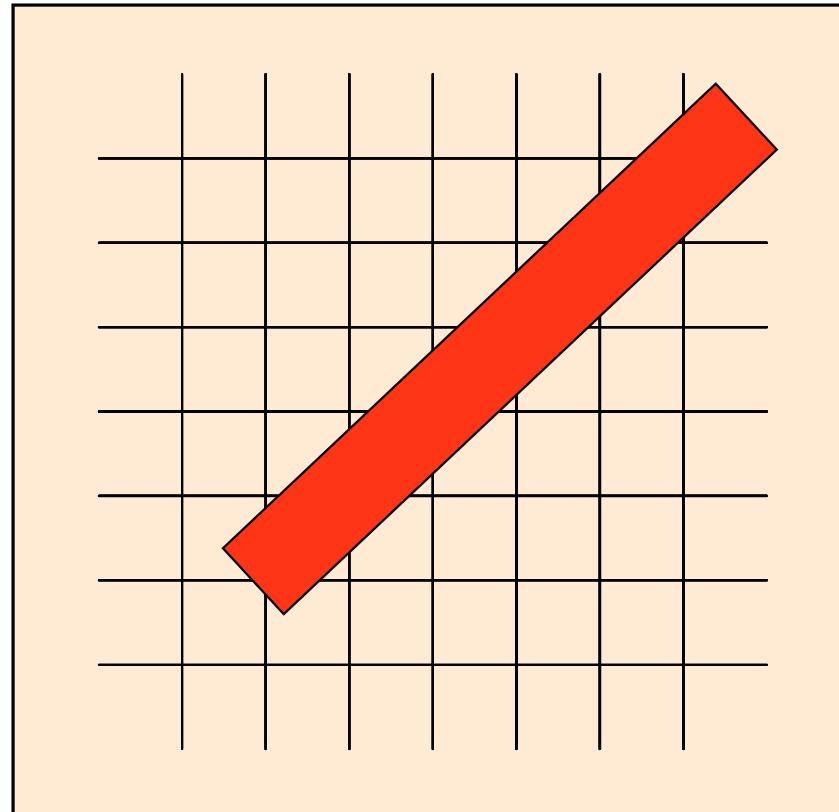
ANTICRÉNELAGE

- Les algorithmes pour les écrans à balayage linéaire produisent des images en escalier



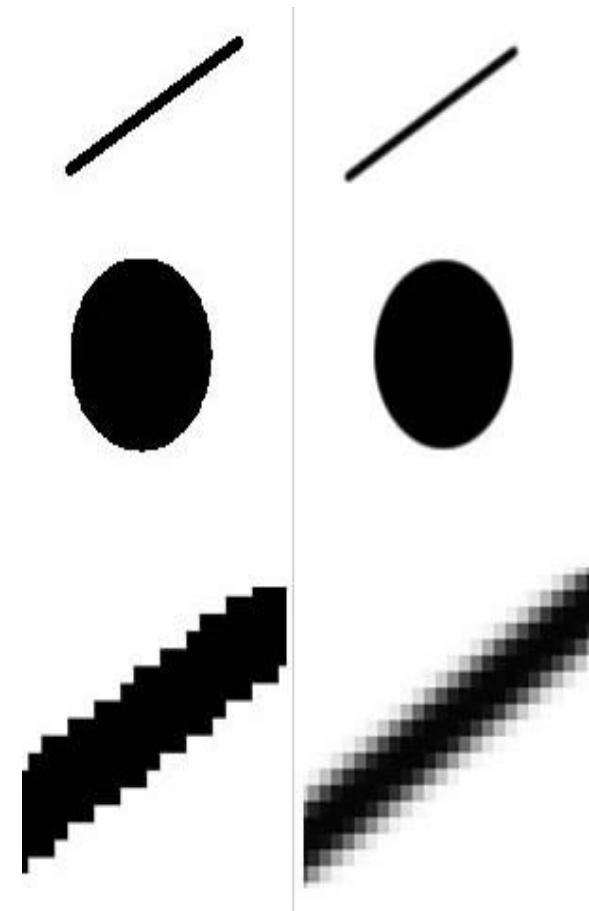
ANTICRÉNELAGE

- ➊ On utilise une plus grande résolution ou on allume les pixels sur le lieu d'une bande imaginaire de largeur finie



ANTICRÉNELAGE

- Le lissage ou anticrénelage (*antialiasing* en anglais) est une technique qui renferme des calculs mathématiques permettant d'éliminer l'effet de crénaux quand on trace des lignes ou des points.
- Un ensemble de crénaux ressemble de très près à une escalier de points formant une ligne ou l'extrémité de la surface d'un polygone. Sans anticrénelage sur une scène dessinée à l'écran d'un ordinateur, il est assez souvent très facile d'apercevoir l'effet de points en escalier et parfois cela gâche la beauté de la scène.



ANTICRÉNELAGE

- ➊ Pour activer l'anticrénelage,

- Activer l'anticrénelage

- ▶ glEnable(**GL_POINT_SMOOTH**);
 - ▶ glEnable(**GL_LINE_SMOOTH**);
 - ▶ glEnable(**GL_POLYGON_SMOOTH**);

- Utiliser la fusion de couleur (*blending*)

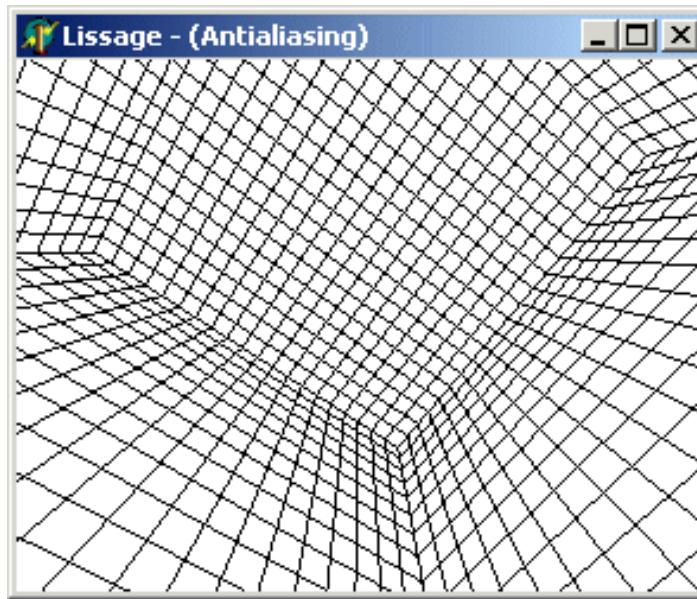
- ▶ glBlendFunc(**GL_SRC_ALPHA**, **GL_ONE_MINUS_SRC_ALPHA**);
 - ▶ glEnable(**GL_BLEND**);

- Désactiver le crénelage

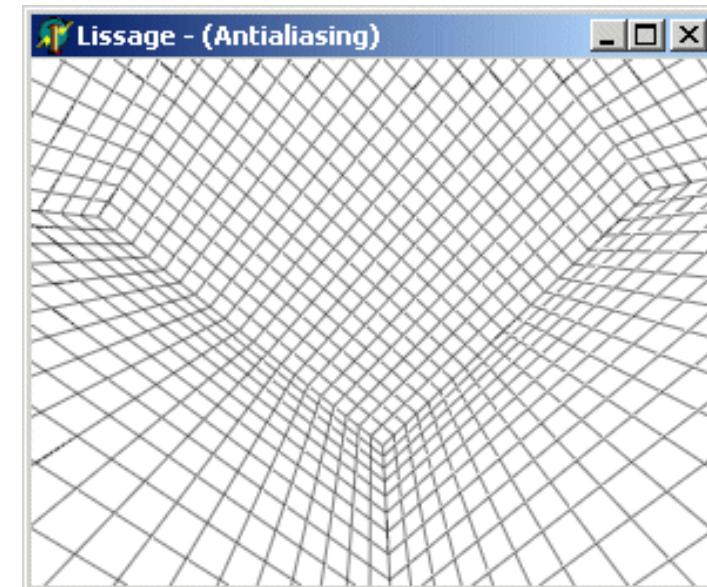
- ▶ glDisable(**GL_POINT_SMOOTH**);
 - ▶ glDisable(**GL_LINE_SMOOTH**);
 - ▶ glDisable(**GL_POLYGON_SMOOTH**);

CRÉNELAGE [Antialiasing]

Avant



Après



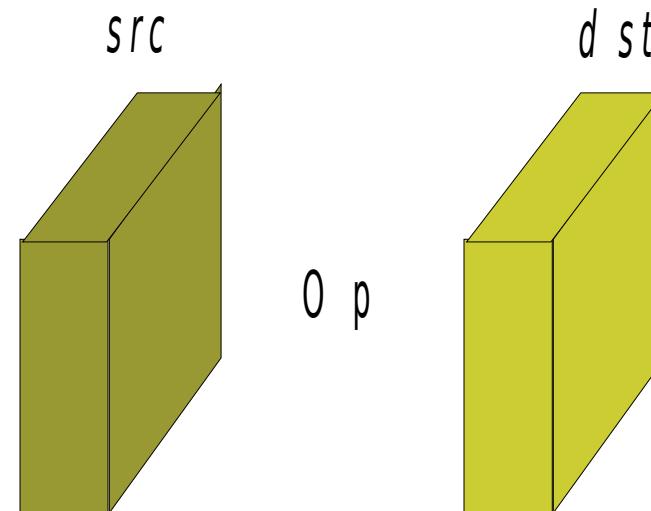
Opérations logiques

Opérations logiques

Comment écrire dans (remplir) la mémoire de trame ?
Utiliser une combinaison logique de pixels.

Quelques combinaisons possibles :

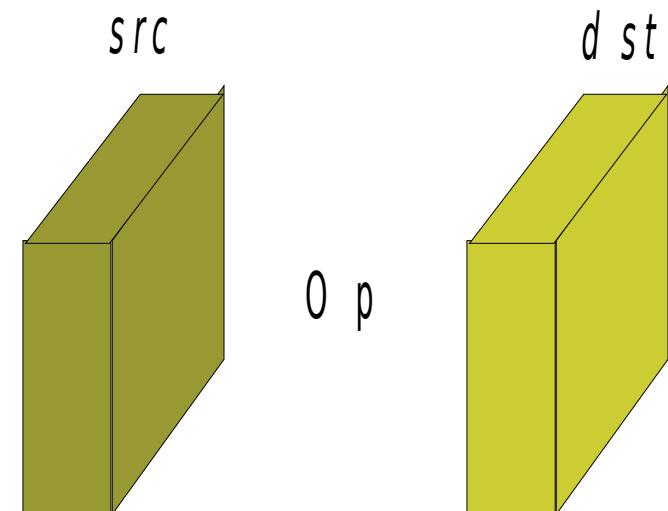
- $dst := src$
- $dst := 0$
- $dst := \text{not } src$
- $dst := dst \text{ or } src$
- $dst := dst \text{ xor } src$
- ...



Opérations logiques

Les 16 opérations/combinaisons disponibles sont :

<code>GL_CLEAR</code>	$dst := 0$
<code>GL_COPY</code>	$dst := src$
<code>GL_NOOP</code>	$dst := dst$
<code>GL_SET</code>	$dst := 1$
<code>GL_COPY_INVERTED</code>	$dst := \text{not } src$
<code>GL_INVERT</code>	$dst := \text{not } dst$
<code>GL_AND_REVERSE</code>	$dst := src \text{ and not } dst$
<code>GL_OR_REVERSE</code>	$dst := src \text{ or not } dst$
<code>GL_AND</code>	$dst := src \text{ and } dst$
<code>GL_OR</code>	$dst := src \text{ or } dst$
<code>GL_NAND</code>	$dst := \text{not} (src \text{ and } dst)$
<code>GL_NOR</code>	$dst := \text{not} (src \text{ or } dst)$
<code>GL_XOR</code>	$dst := src \text{ xor } dst$
<code>GL_EQUIV</code>	$dst := \text{not} (src \text{ xor } dst)$
<code>GL_AND_INVERTED</code>	$dst := \text{not } src \text{ and } dst$
<code>GL_OR_INVERTED</code>	$dst := \text{not } src \text{ or } dst$



Opérations logiques

Le choix de la combinaison est fait par :

```
void glLogicOp( GLenum opcode );  
où opcode spécifie la combinaison
```

- GL_COPY est la combinaison par défaut

Opérations logiques

Les variables d'état booléennes GL_COLOR_LOGIC_OP et GL_INDEX_LOGIC_OP servent à activer cette fonctionnalité :

- Activer :

```
glEnable( GL_COLOR_LOGIC_OP );  
glEnable( GL_INDEX_LOGIC_OP );
```

- Désactiver :

```
glDisable( GL_COLOR_LOGIC_OP );
```

Opérations logiques

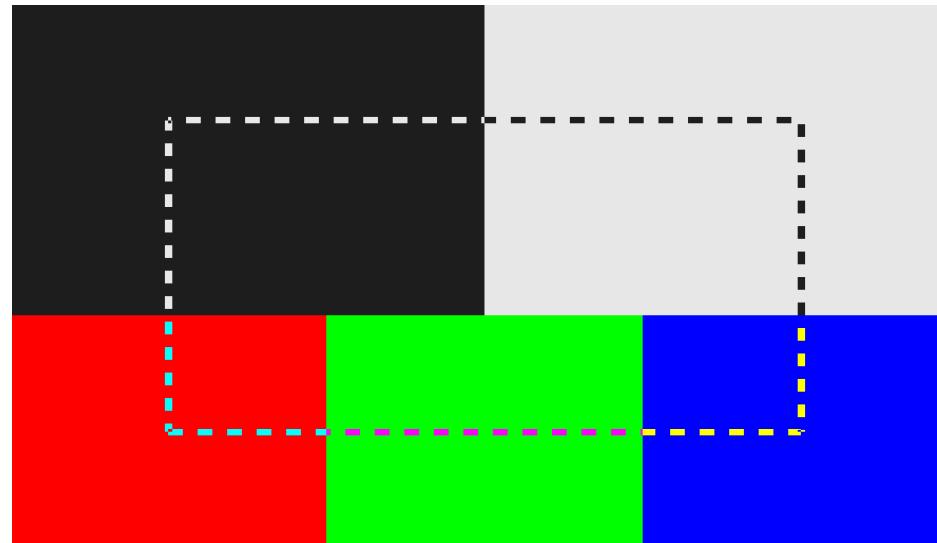
Exemple:

```
glLogicOp( GL_XOR );
 glEnable( GL_COLOR_LOGIC_OP );
 glColor3f( 1., 1., 1. );
 glBegin( GL_LINE_STRIP );
 glVertex2i(5,5);
 glVertex2i(10,7);
 glVertex2i(3,15);
 glEnd();
 glDisable( GL_COLOR_LOGIC_OP );
```

Opérations logiques

Tracer un rectangle élastique

- Tracer un rectangle élastique en utilisant les opérations logiques
- Peut servir à définir une région d'intérêt
- Inverse automatiquement les couleurs des pixels si on utilise l'opération GL_XOR :



Opérations logiques

Tracer un rectangle élastique

- Tracer un rectangle élastique en trois temps :
 - 1- l'initialisation
 - 2- le retraçage en cours de manipulation par l'usager
 - 3- la remise à l'état normal
- L'idée est de modifier la transformation virtuelle à affichage de façon à ce que *une unité virtuelle = un pixel*.
- On évite ainsi de transformer de virtuel à affichage et vice-versa pour tracer le rectangle.
- La plus grande partie du code d'initialisation sert à changer temporairement la transformation.

Opérations logiques

Tracer un rectangle élastique : 1-initialisation

```
courant = ancrage = ev->pos(); // position des deux coins du rectangle

// Sauvegarder les attributs de traçage
glPushAttrib( GL_LINE_BIT | GL_CURRENT_BIT | GL_COLOR_BUFFER_BIT | GL_ENABLE_BIT );
// Sauvegarder les transformations courantes
glMatrixMode( GL_MODELVIEW ); glPushMatrix(); glLoadIdentity();
glMatrixMode( GL_PROJECTION ); glPushMatrix(); glLoadIdentity();
// Initialiser la matrice 1 pixel = 1 unité virtuelle
GLint Cloture[4]; glGetIntegerv( GL_VIEWPORT, Cloture );
gluOrtho2D( 0.0, (GLdouble) Cloture[2] , (GLdouble) Cloture[3], 0.0 );
glMatrixMode( GL_MODELVIEW );

// Changer le mode de traçage des rectangles, initialiser la couleur et le mode XOR
glPolygonMode( GL_FRONT_AND_BACK, GL_LINE );
glColor3f( 1.0, 1.0, 1.0 );
 glEnable( GL_COLOR_LOGIC_OP );
 glLogicOp( GL_XOR );
 glDisable( GL_DEPTH_TEST );

// On dessinera dans le tampon déjà visible à l'écran
GLint drawBuffer; glGetIntegerv( GL_DRAW_BUFFER, &drawBuffer );
glDrawBuffer( GL_FRONT );

// Tracer un premier rectangle (un seul point)
glRectf( ancrage.x(), ancrage.y(), courant.x(), courant.y() );
```

Opérations logiques

Tracer un rectangle élastique : 2-retraçage en cours de ...

```
// Tracer l'ancien rectangle (ça efface l'ancien rectangle à cause du XOR)
glRectf( ancrage.x(), ancrage.y(), courant.x(), courant.y() );

// Récupérer la nouvelle position et tracer le nouveau rectangle
courant = ev->pos();
glRectf( ancrage.x(), ancrage.y(), courant.x(), courant.y() );
```

Opérations logiques

Tracer un rectangle élastique : 3-remise à l'état normal

```
// Tracer l'ancien rectangle (ça efface l'ancien rectangle à cause du XOR)
glRectf( ancrage.x(), ancrage.y(), courant.x(), courant.y() );

// Désactiver les opérations bitblt
glDisable( GL_COLOR_LOGIC_OP );

// On dessinera dans le tampon habituel
glDrawBuffer( drawBuffer );

// Rétablir les transformations initiales
glMatrixMode( GL_PROJECTION );
glPopMatrix();
glMatrixMode( GL_MODELVIEW );
glPopMatrix();

// Remettre les attributs initiaux
glPopAttrib();
```

Utilisation du stencil

Utilisation du stencil

- On utilise le stencil comme un pochoir pour masquer l'affichage de certaines régions
- L'affichage est conditionnel:
 - aux valeurs présentes dans un tampon de stencil
 - au test de comparaison choisi (une fonction)
- Le tampon du stencil contient des entiers (un ou plusieurs bits par pixel)
- On « dessine » dans le tampon du stencil pour le remplir
- Même si on y dessine, on ne voit pas directement ce tampon!

Utilisation du stencil

- Le test compare la valeur du tampon de stencil à l'emplacement du fragment et la valeur de référence courante du stencil
- Si le test de stencil échoue, le fragment est oublié (*discard*), et la couleur ainsi que la profondeur demeurent inchangés
- Dans tous les cas, l'opération associée à un échec du test, ou à un succès, est appliquée à la valeur du stencil
- Le stencil effectue donc deux opérations:
 - vérifie si le fragment poursuit vers le tampon d'affichage
 - met à jour le stencil en fonction du résultat du test

Utilisation du stencil

```
void glStencilFunc( GLenum fonction,  
                    GLint référence, GLuint masque );
```

Quelques valeurs que peut prendre **fonction** :

GL_ALWAYS: Le test réussit toujours (par défaut).

GL_NEVER: Le test ne réussit jamais.

GL_EQUAL: Le test réussit si la valeur de référence est égale à celle du stencil.

(**GL_LESS**, **GL_LEQUAL**, **GL_GREATER**, **GL_GEQUAL**, **GL_NOTEQUAL**)

Le masque est un masque de bits qui est appliqué à la référence *et* à la valeur de comparaison du stencil.

Le test du stencil est : (ref & masque) fonction (stencil & mask)

Exemple :

```
glStencilFunc( GL_ALWAYS, 1 /*référence*/, 1 /*masque*/ );
```

(ici, le test réussit toujours, avec une référence de 1 et un masque de 1)

Utilisation du stencil

```
void glStencilOp(  
    GLenum fail, // Le test du stencil échoue  
    GLenum zfail, // Le test du stencil réussit, mais celui de profondeur échoue  
    GLenum pass ); // Les deux tests passent (ou il n'y pas de test de profondeur)
```

En fonction de l'opérateur, le stencil peut prendre différentes valeurs:

GL_KEEP : garde la valeur courante dans le stencil

GL_ZERO : met la valeur du stencil à zéro

GL_REPLACE : remplace la valeur du stencil par la valeur de référence

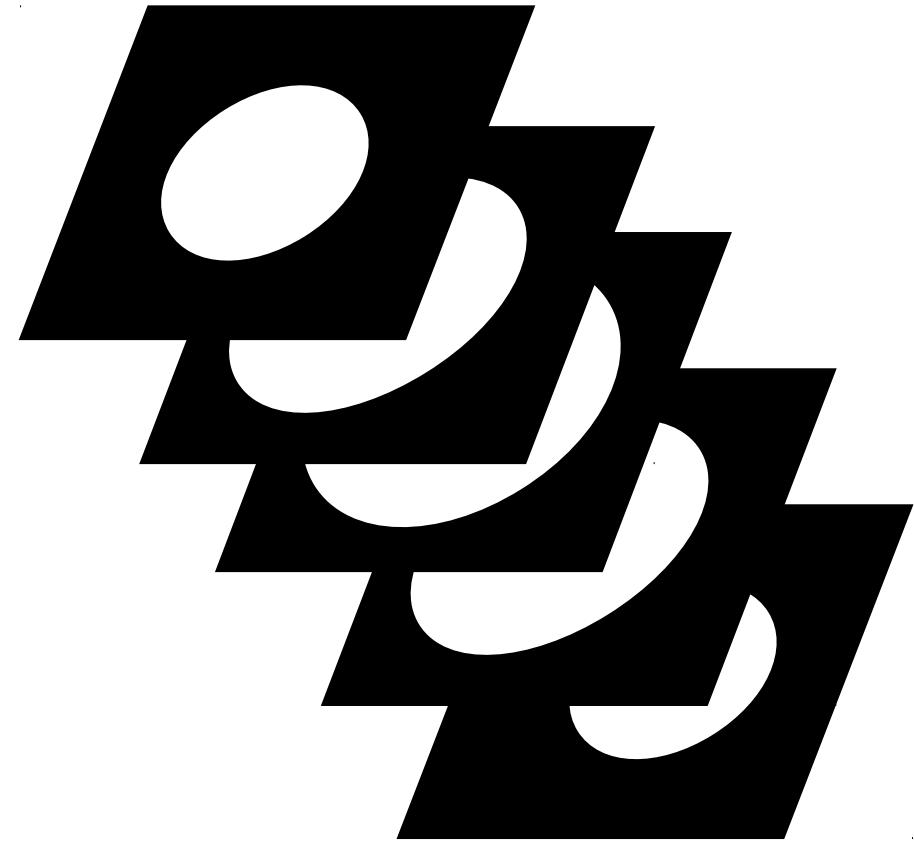
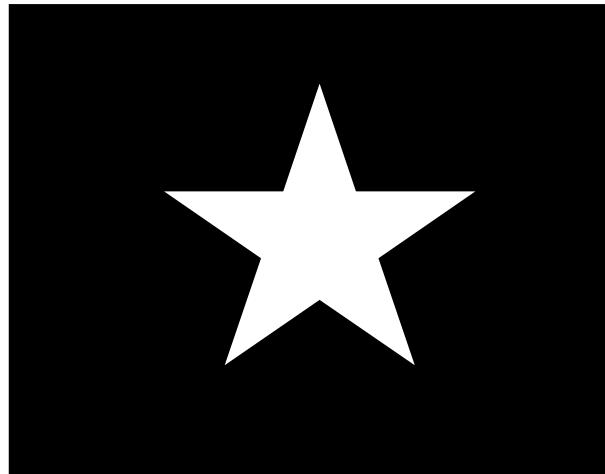
(celle spécifiée par glStencilfunc())

GL_INCR : incrémente la valeur du stencil

GL_DECR : décrémente la valeur du stencil

GL_INVERT : fait un complément à un de la valeur du stencil

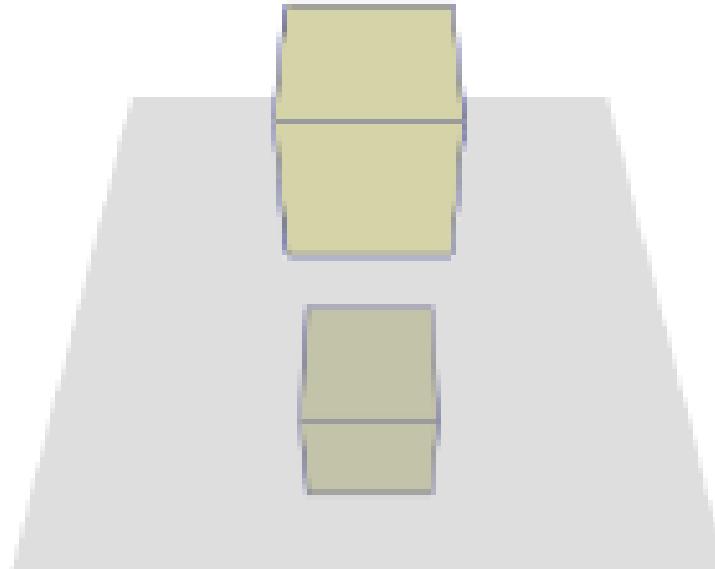
Des exemples de stencil....



Attention! Le stencil est composé de plans de bits
(de 0 à $2^n - 1$, tel que spécifié par ref). Ceci nous permet
même d'y « dessiner » des géométries tridimensionnelles ...

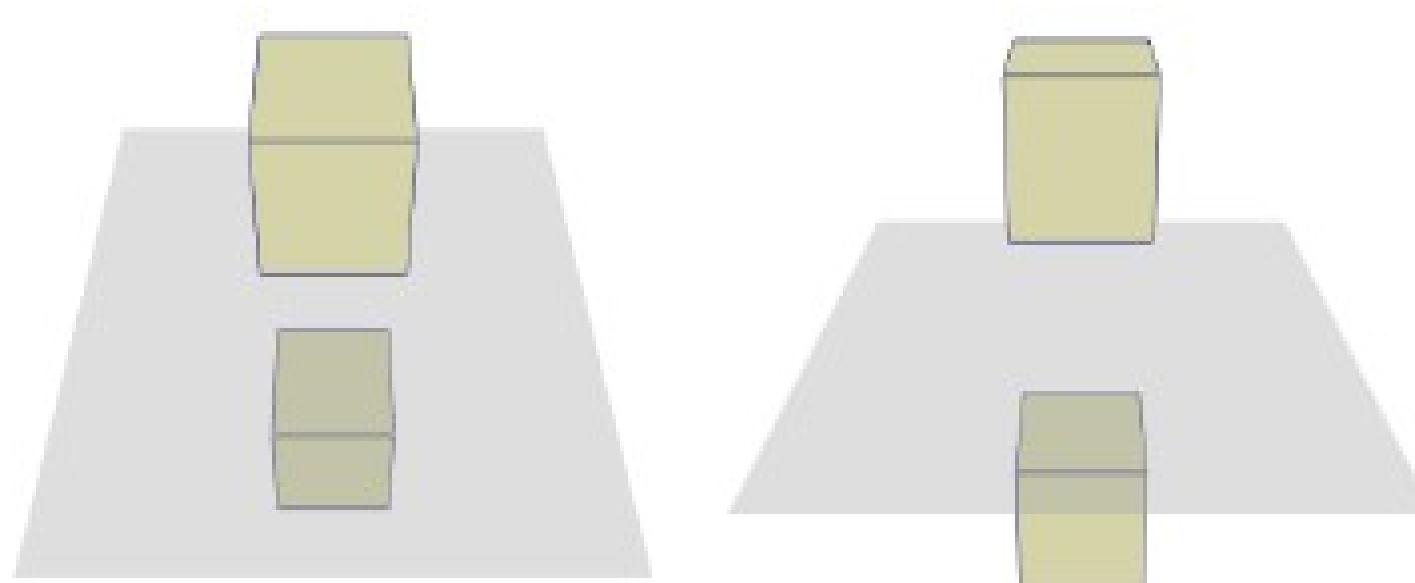
Utilisation du stencil pour un miroir

- 1) On dessine la réflexion de l'objet avec `glScalef(1, -1, 1);`
- 2) On dessine le véritable objet
- 3) On dessine le miroir



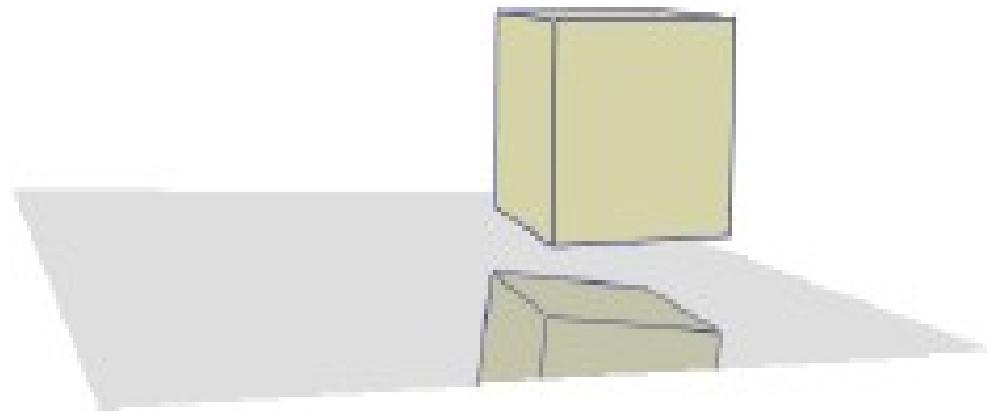
Utilisation du stencil pour un miroir

- Un problème survient si l'objet déborde la surface du miroir
- On souhaiterait alors limiter la réflexion à la surface du miroir



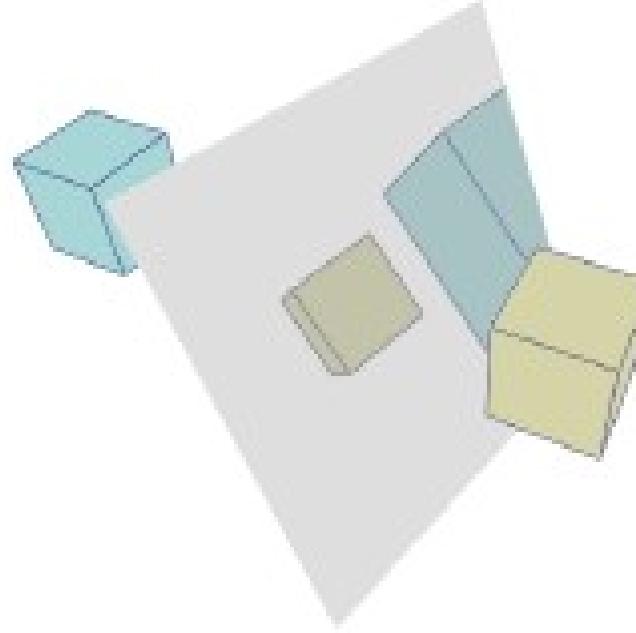
Utilisation du stencil pour un miroir

- On peut utiliser le stencil comme pochoir
- On écrit 1 dans le stencil pour tous les fragments qui correspondent à la partie au miroir et 0 ailleurs
- On dessine la réflexion en utilisant le test du stencil



Utilisation du stencil pour un miroir

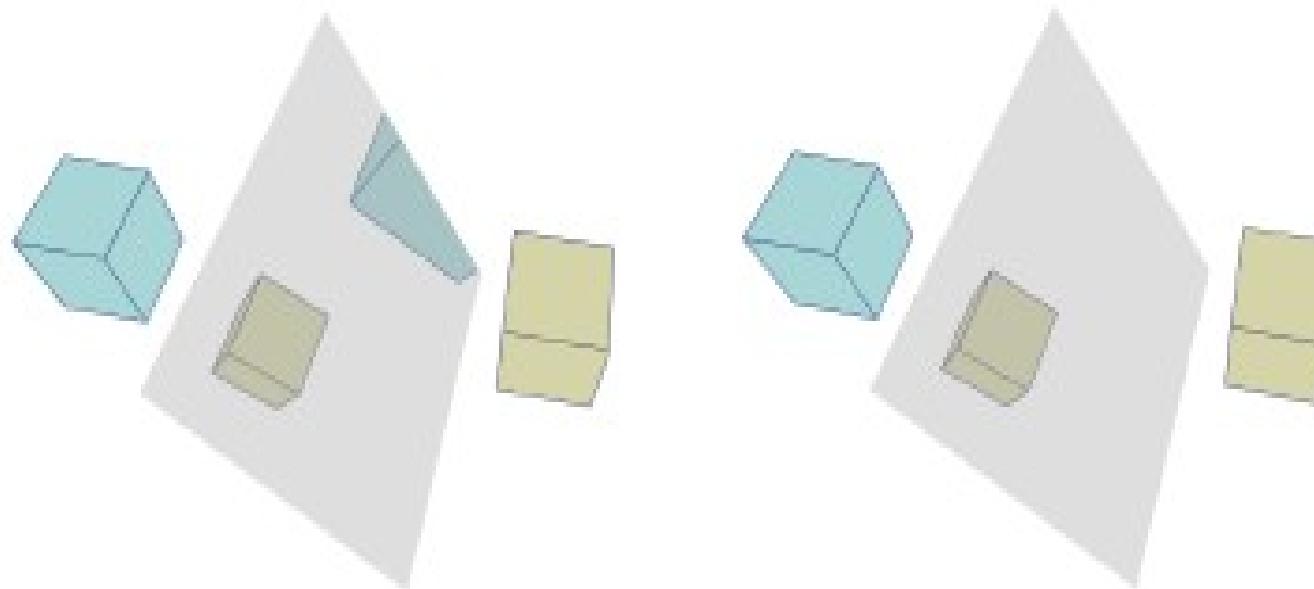
- Un second problème peut survenir : les objets situés derrière le miroir sont transformés par symétrie et ils s'affichent devant le miroir
- Par exemple, ici, le cube bleu est en arrière du miroir, mais apparaît ici (incorrectement) aussi devant le miroir :



Supprimer la réflexion de ces objets

Utilisation du stencil pour un miroir

- La solution est d'ajouter un plan de découpage qui coupe ce qui d'un côté du plan.
- On choisit le plan du miroir pour découper.



Utilisation du stencil pour un miroir

Plan de découpage :

```
void glClipPlane( GLenum plan, GLdouble *equation );
```

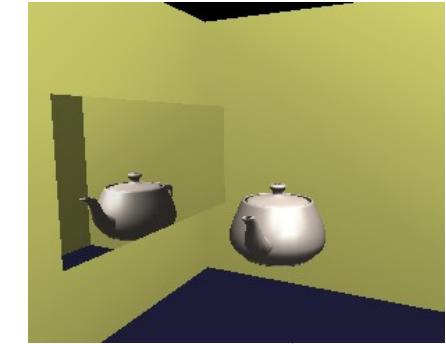
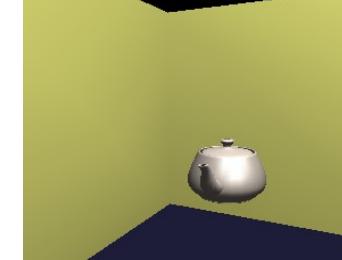
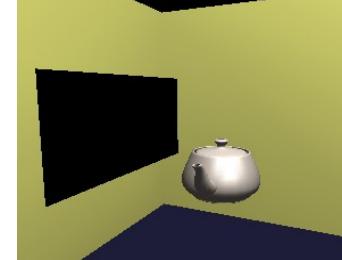
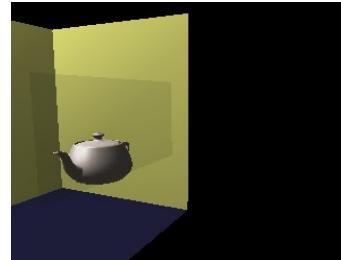
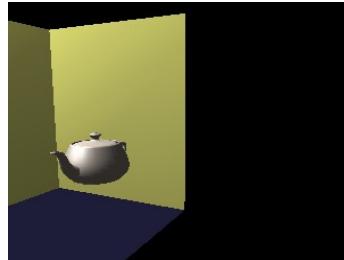
plan : un entier entre 0 et GL_MAX_CLIP_PLANES-1

equation : un tableau de 4 valeurs donnant l'équation du plan.

Ce sont les valeurs A,B,C,D de l'équation du plan $Ax+By+Cz+D=0$.

Utilisation du stencil pour un miroir

- Exemple plus complet : on utilise le stencil comme un pochoir pour masquer l'affichage de certaines régions
- Ce qu'on trace:
 - 1) Remplir le stencil avec le miroir
 - 2) Tracer la scène réfléchie
 - 3) Tracer le miroir (semi-transparent)
 - 4) Tracer la scène normalement



Utilisation du stencil pour un miroir

```
void Dessiner() // pour un miroir
{
    glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT | GL_STENCIL_BUFFER_BIT );
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity();

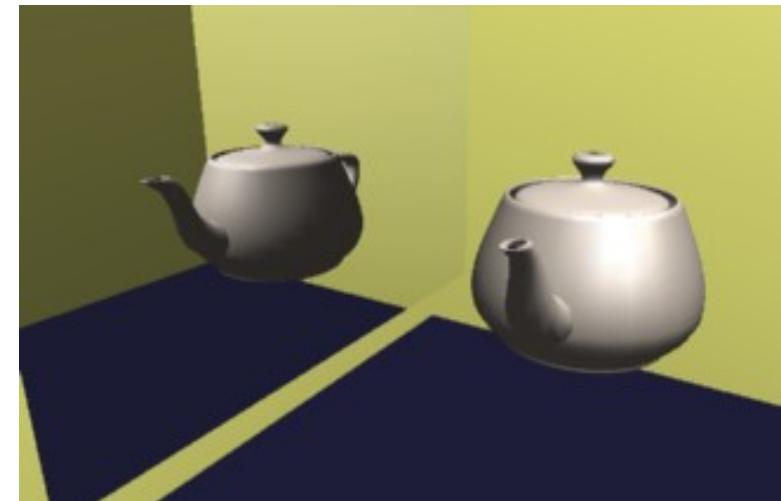
    gluLookAt( cameraX, cameraY, cameraX,
               oeilX, oeilY, oeilZ, hautX, hautY, hautZ );

    glEnable( GL_STENCIL_TEST );
    // Ces deux opérations font que les fragments n'arriveront pas à l'écran,
    // mais le tampon de stencil se remplira de '1' aux endroits correspondants
    glStencilFunc( GL_NEVER, 1, 1 );
    glStencilOp( GL_REPLACE, GL_REPLACE, GL_REPLACE );

    // on dessine le miroir dans le tampon du stencil
    DessinerMiroir( );
```

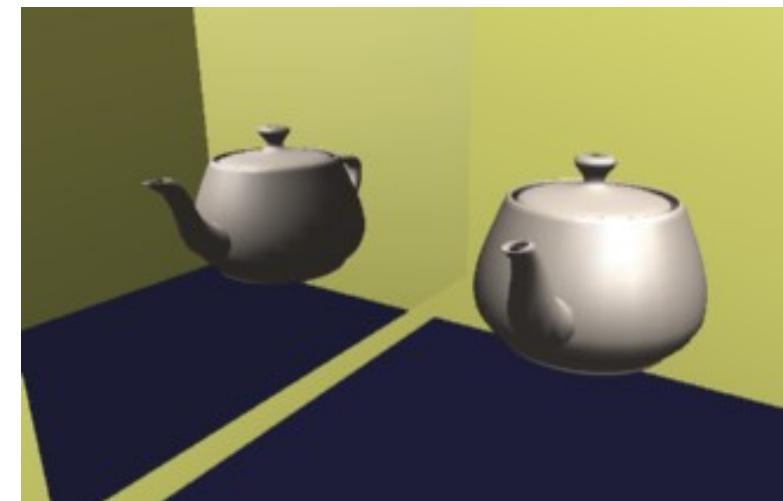
Utilisation du stencil pour un miroir

```
// Ces deux opérations font qu'on ne dessine que là où  
// le tampon de stencil est à '1'  
glStencilFunc( GL_EQUAL, 1, 1 );  
glStencilOp( GL_KEEP, GL_KEEP, GL_KEEP );  
  
// dessiner la réflexion d'abord  
glPushMatrix() ;  
glScalef( 1.0, -1.0, 1.0 );  
  
DessinerSceneReflechie();  
  
glPopMatrix();
```



Utilisation du stencil pour un miroir

```
glDisable( GL_STENCIL_TEST );  
  
// il faut dessiner le miroir avec du blending pour voir la réflexion  
glEnable( GL_BLEND );  
glBlendFunc( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );  
DessinerMiroir();  
glDisable( GL_BLEND );  
  
DessinerScene();  
}
```



Utilisation du stencil pour un miroir

Plein de choses à faire avec le stencil:

- Des réflexions
- De la géométrie constructive
- Des décalques
- Composition d'images en profondeur
- etc.



Brouillard

BROUILLARD

Effets atmosphériques

- Simulations d'effets atmosphériques: brouillard, pollution, etc.
- La couleur du brouillard est mélangée à la couleur du fragment
- Ceci permet de limiter la visibilité selon la profondeur
- Un effet visuel efficace avec un coût minime!

Exemple :

```
glEnable( GL_FOG );
glFog*( ... ); // couleur, début et fin de la zone
// autres modes : GL_FASTEST ou GL_DONT_CARE
glHint( GL_FOG_HINT, GL_NICEST );
```

FACTEUR BROUILLARD

Effets atmosphériques

Contrôle de la densité:

- Exponentielle (GL_EXP)

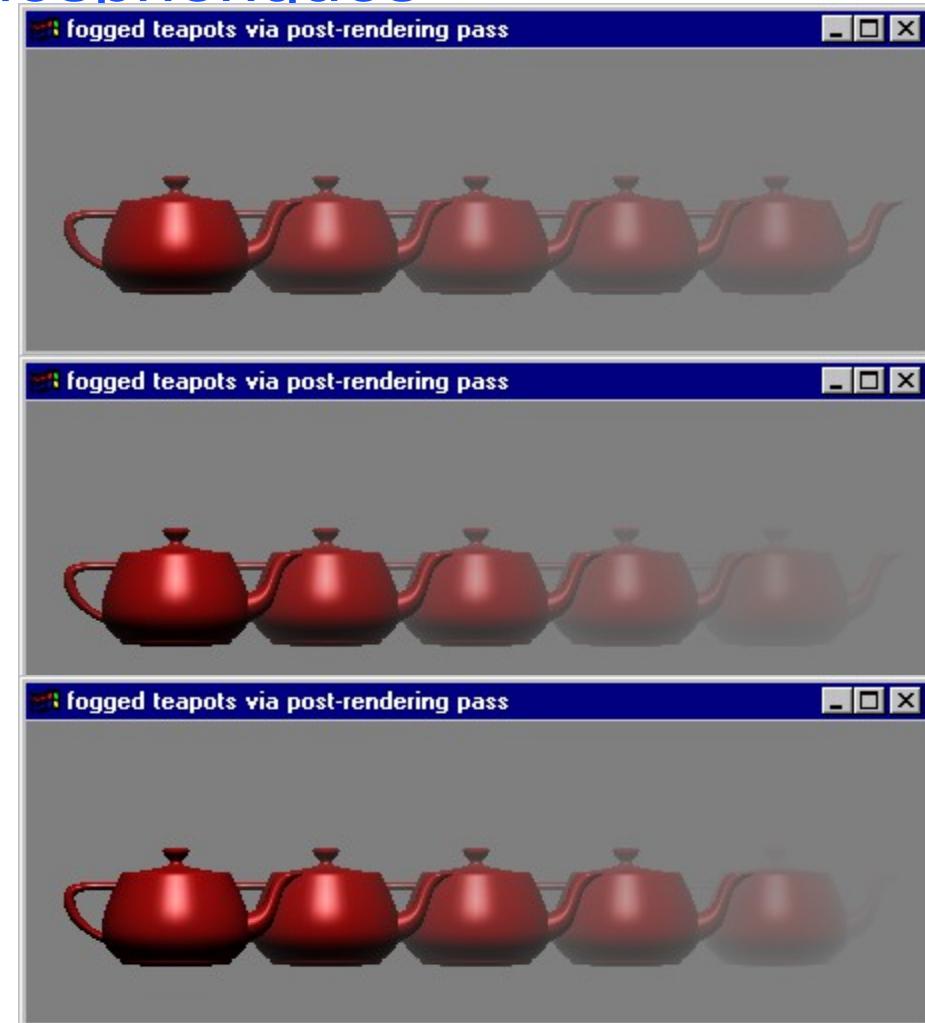
$$f = e^{-(\text{densité}-z)}$$

- Exponentielle 2 (GL_EXP2)

$$f = e^{-(\text{densité}-z)^2}$$

- Linéaire (GL_LINEAR)

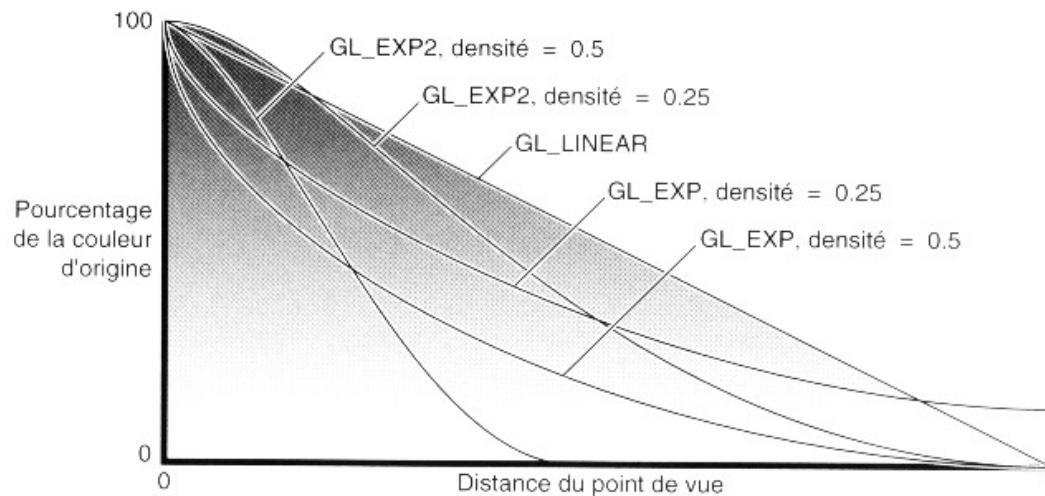
$$f = \frac{\text{fin} - z}{\text{fin} - \text{début}}$$



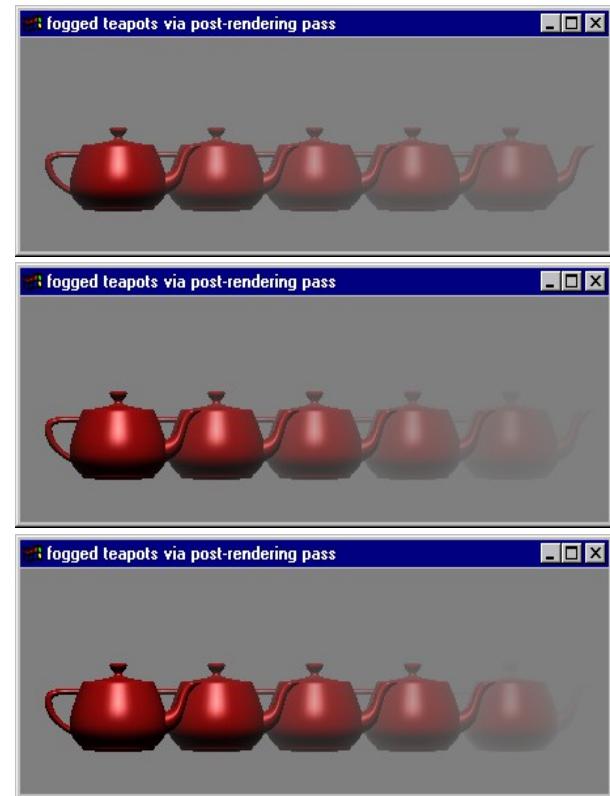
FACTEUR BROUILLARD

Effets atmosphériques

Quelques exemples de courbes de densité:



(Cette image est extraite de l'ouvrage OpenGL 1.2, Woo, Neider, Davis & Shreiner chez CampusPress.)



FACTEUR BROUILLARD

Effets atmosphériques

- ➊ z : généralement la profondeur (z) du fragment
(ou une autre valeur alors spécifiée par *glFogCoord*)
début : début de la courbe pour le calcul du facteur
fin : fin de la courbe pour le calcul du facteur
- ➋ f est utilisé pour calculer la couleur résultante due au brouillard
Couleur = $f C_i + (1-f) C_f$
 C_i : couleur du fragment
 C_f : couleur du brouillard

FACTEUR BROUILLARD

Effets atmosphériques

```
void glFog{if}( GLenum pname, TYPE param );  
void glFog{if}v( GLenum pname, TYPE lparam );
```

Paramètre	Valeur par défaut	Signification
GL_FOG_MODE	GL_EXP	GL_EXP, GL_EXP2, GL_LINEAR
GL_FOG_COLOR	(0.0, 0.0, 0.0, 0.0)	Couleur du brouillard
GL_FOG_INDEX	0	Index de couleur
GL_FOG_DENSITY	1.0	Densité pour le calcul du facteur
GL_FOG_START	0.0	Début pour le calcul du brouillard
GL_FOG_END	1.0	Fin pour le calcul du brouillard

```
void glEnable( GL_FOG );  
void glDisable( GL_FOG );
```