

Corso di
Informatica Grafica

Introduzione
a
OpenGL

parte 5

Componenti della luce

I colori degli oggetti percepiti dall'occhio umano sono il risultato del fenomeno di riflessione della luce. Un oggetto illuminato da una sorgente di luce bianca appare rosso se riflette la sola componente rossa della luce incidente e assorbe tutte le altre componenti.

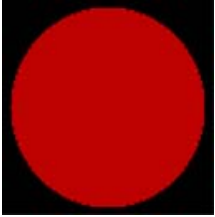
Per ragioni di efficienza OpenGL possiede un meccanismo di calcolo della luce semplificato che non tiene conto di molti aspetti della meccanica della luce.

Il modello di illuminazione di cui dispone OpenGL prevede che la luce abbia tre componenti:

- Ambient light : luce dell'ambiente, che arriva da tutte le direzioni poiché è stata riflessa numerose volte da più superfici
- Diffuse light : luce che arriva da una direzione preferenziale, ma che viene riflessa equamente in tutte le direzioni quando incide su una superficie
- Specular light: come la luce diffusa arriva da una direzione precisa, ma è riflessa in una sola direzione (riproduce bene l'aspetto di un oggetto metallico)

Componenti della luce

Esempi:



solo luce ambientale rossa: i contorni 3d della sfera non possono essere distinti



solo luce diffusa: luce direttiva “rossa” che illumina una sfera e che viene completamente riflessa



luce ambientale + luce diffusa



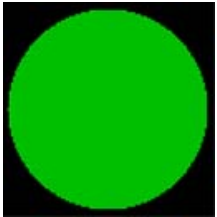
luce ambientale + luce diffusa + luce speculare

La luce speculare viene riflessa in una sola direzione

(effetto tipico di materiali metallici)

Componenti della luce

Esempi:



Sfera con una componente di luce emissiva “verde”. Effetto simile all'illuminazione con luce ambientale



Se la sfera verde viene illuminata con una luce rossa con componenti ambientale, diffusa e speculare la sfera viene percepita come gialla

Componenti della luce

Come visto negli esempi precedenti ciascuna sorgente luminosa è tipicamente formata da una sovrapposizione di luce ambientale, diffusa e speculare, anche se può esserci un tipo di luce prevalente che ne determina le proprietà. Le componenti di una sorgente di luce si specificano esattamente come i colori, con le componenti rossa, verde e blu : ad esempio, per un raggio laser (luce rossa) le componenti potrebbero essere:

	Red	Green	Blue	Alpha
Specular	0.99	0	0	1
Diffuse	0.1	0	0	1
Ambient	0.05	0	0	1

Abilitare le luci

L'illuminazione va abilitata on **glEnable(GL_LIGHTING)**. Se non si specificano le sorgenti di luce e le proprietà dei materiali, da sola questa chiamata rende la scena completamente buia.

Successivamente è possibile indicare un valore globale di illuminazione ambientale che colpisce tutti gli oggetti della scena utilizzando la funzione

void glLightModelfv(*GLenum pname, GLfloat *params*)

in cui *pname* vale `GL_LIGHT_MODEL_AMBIENT` e attiva la luce d'ambiente con le componenti specificate da *params*.

Per disabilitare l'illuminazione ambientale impostare *params* con tutti i valori a 0.

Esempio:

```
GLfloat global_ambient[] = { 0.5f, 0.5f, 0.5f, 1.0f };
```

```
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient);
```

Accendere le luci

Tipicamente le proprietà delle luci si definiscono nella funzione di setup del rendering context (SetupRC nei nostri esempi) in due passi:

- Si costruisce un array di 4 elementi con le proprietà della luce
- Si chiama la funzione

`void glLightfv(GLenum light, GLenum pname, const GLfloat *params)`

in cui *light* può valere GL_LIGHT0, GL_LIGHT1...fino a GL_MAXLIGHT, *pname* specifica che componente si modifica: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSIVE ... e *params* contiene i nuovi valori.

- Si accende la luce con `glEnable(GL_LIGHTx)` con x=0,1,2...

E' possibile disabilitare tutte le luci (utile quando si vuole disegnare oggetti che non devono essere influenzati dalle luci) con il comando `glDisable(GL_LIGHTING)`. E' anche possibile disabilitare una sorgente di luce particolare con il comando `glDisable(GL_LIGHTx)` con x=0,1,2...

Posizionare le luci

La luce può essere posizionata in un punto a piacere utilizzando sempre la funzione `glLightfv`, con `pname=GL_POSITION` e `params` contenente l'indirizzo del vettore posizione (possibile collocazione del codice per il posizionamento delle luci è in una funzione di inizializzazione.)

Nel posizionare la luce, se l'ultimo elemento dell'array è diverso da zero significa che i tre elementi precedenti costituiscono le coordinate della posizione della luce (**luce posizionale**), mentre se l'ultimo valore è uguale a 0 allora la luce è a distanza infinita dalla scena e i tre valori nell'array indicano la sua direzione (**luce direzionale**). Nel caso di luce direzionale il vettore direzione punta verso la direzione di provenienza della luce. Per esempio una direzione (0,0,1,0) significa una luce direzionale lungo l'asse Z negativo (-Z).

Osservazione importante per disegnare luci dinamiche, cioè luci che modificano la loro direzione di propagazione: **la posizione della luce viene trasformata dalla matrice di MODELVIEW corrente.**

Es: `GLfloat position[] = { -1.5f, 1.0f, -4.0f, 1.0f };`

`glLightfv(GL_LIGHT0, GL_POSITION, position);`

Attenuazione delle luci

L'intensità delle luci posizionali può essere attenuata dalla distanza sorgente-oggetto. Nello stato di default l'attenuazione è disabilitata.

Il fattore di attenuazione è definito come $\text{attenuation factor} = \frac{1}{k_c + k_l d + k_q d^2}$

d = distance between the light's position and the vertex

k_c = GL_CONSTANT_ATTENUATION (default = 1)

k_l = GL_LINEAR_ATTENUATION (default = 0)

k_q = GL_QUADRATIC_ATTENUATION (default = 0)

quindi di default l'attenuazione non viene considerata

E' possibile modificare i parametri di attenuazione come nel seguente esempio:

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);  
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);  
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Osservazione: le luci direzionali non vengono attenuate

Calcolo della componente speculare

Come sappiamo la componente di riflessione speculare dipende dalla posizione dell'osservatore, dalla normale alla superficie e dalla posizione della sorgente di luce.

Per ragioni di efficienza, OpenGL di default assume che il vettore dal vertice all'osservatore (vertex-to-eye) sia costante e pari a (0,0,1). Questa è una forte semplificazione.

Per abilitare il calcolo più realistico delle riflessioni speculari è necessario invocare la seguente funzione (che nei nostri esercizi di solito non compare)

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

Proprietà dei materiali

Per definire in modo completo il modello di illuminazione di una scena oltre alle luci occorre specificare le proprietà dei materiali. OpenGL infatti calcola il colore finale di un oggetto combinando le informazioni sulle luci incidenti con le proprietà dei materiali. Per specificare le proprietà di illuminazione dei materiali si specifica come tale oggetto riflette le componenti RGB della luce che lo illumina, ad esempio un oggetto metallico potrebbe riflettere la maggior parte della luce speculare e assorbire maggiormente quella diffusa. La funzione da utilizzare per specificare le proprietà dei materiali è

```
void glMaterialfv(GLenum face, GLenum pname, const GLfloat *params)
```

in cui *face* può valere GL_FRONT, GL_BACK o GL_FRONT_AND_BACK, *pname* è la proprietà del materiale e può valere GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_AMBIENT_AND_DIFFUSE ..., infine *params* contiene l'indirizzo di un array contenente i valori da assegnare alla proprietà.

Esempio:

```
float mcolor[] = { 1.0f, 0.0f, 0.0f, 1.0f }; // l'oggetto  
glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mcolor); // riflette luce rossa
```

Il parametro GL_AMBIENT_AND_DIFFUSE assegna lo stesso colore alle componenti ambientali e diffuse del materiale. Si utilizza spesso perché di solito le componenti ambientali e diffuse dei materiali si impostano uguali.

Proprietà dei materiali

Osservazione importante:

Quando si assegna un colore ad un oggetto con il comando glColor (come siamo abituati a fare) **non** si sta assegnando nessuna proprietà del materiale. Il colore che si assegna all'oggetto con glColor non reagisce con la luce incidente. I modi corretti per gestire i colori degli oggetti e le luci sono quindi 3:

- 1) Assegnare i colori agli oggetti con glColor con l'illuminazione disabilitata (come siamo abituati a fare). Questo e' il caso piu' semplice ma che preclude l'uso dell'illuminazione e quindi delle ombreggiature (shading) delle superfici.
- 2) Assegnare le proprietà dei materiali con **glMaterialfv** per ciascun oggetto con l'illuminazione abilitata.
- 3) Assegnare i colori agli oggetti con glColor con l'illuminazione abilitata in modalità “color tracking”. E' utile per evitare di dover specificare manualmente le proprietà dei materiali di tutti gli oggetti, cosa che puo' essere noiosa se non sono necessari effetti luminosi particolari.

Proprietà dei materiali

La modalità “color tracking” setta automaticamente le proprietà del materiale basandosi sul colore corrente: tale caratteristica va abilitata con

glEnable(GL_COLOR_MATERIAL), quindi con

void glColorMaterial(*GLenum face*, *GLenum mode*)

Es:

```
// da qualche parte nel codice di inizializzazione
```

```
glEnable(GL_COLOR_MATERIAL);
```

```
// ogni successiva chiamata a glColor assegnerà le proprietà ambientali e diffusive dell'oggetto pari ai valori di  
// impostati con glColor
```

```
glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
```

```
// ... da qualche parte nella procedura di rendering ...
```

```
glColor3f(0.0f, 0.0f, 1.0f); // proprietà di riflessione “blu”
```

```
glBegin(GL_TRIANGLES);
```

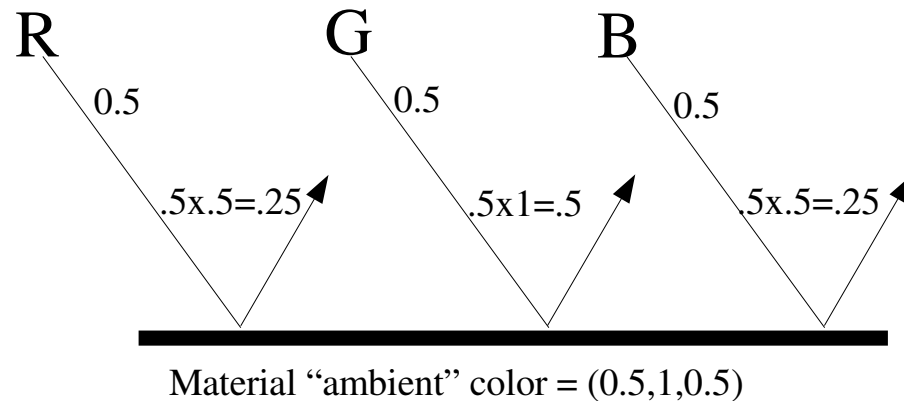
```
glVertex3f(-1.0f, 0.0f, 0.0f); glVertex3f(0.0f, -1.0f, 0.0f); glVertex3f(1.0f, 0.0f, 0.0f);
```

```
glEnd();
```

Effetti della luce

- Luce ambientale

le componenti della luce vanno moltiplicate per quelle della proprietà riflessiva dell'oggetto illuminato per ottenere il “vero” colore



- Luce diffusa

per questa componente l'effetto dipende anche dalla direzione con cui incide sulla superficie dell'oggetto. Di solito le sue componenti RGB sono le stesse della luce ambientale

- Luce speculare

tende ad essere bianca. La sua intensità dipende fortemente dall'angolo di incidenza e i riflessi sono solitamente bianchi (tipica degli oggetti metallici). Può essere associata anche ad una componente di LUCIDITA' definibile, ad esempio, con il seguente comando:

```
glMaterial(GL_FRONT, GL_SHININESS, 128);
```

Equazione di illuminazione

Il colore finale di un vertice (in modalità RGBA) viene calcolato utilizzando una equazione di illuminazione che comprende tre termini: luce emessa dal materiale, prodotto della componente di riflessione ambientale del materiale per la componente di luce ambientale globale e contributo di ogni singola sorgente di luce abilitata. Un'espressione semplificata è la seguente:

$$\begin{aligned} \text{vertex color} = & \text{emission}_{\text{material}} + \text{ambient}_{\text{light_model}} * \text{ambient}_{\text{material}} + \\ & + \sum_{\text{luci}} (\text{att}) \left[\text{ambient}_{\text{light}} * \text{ambient}_{\text{material}} + \right. \\ & \quad + (\mathbf{l} \cdot \mathbf{n}) * \text{diffuse}_{\text{light}} * \text{diffuse}_{\text{material}} + \\ & \quad \left. + (\mathbf{h} \cdot \mathbf{n})^{\text{shininess}} * \text{specular}_{\text{light}} * \text{specular}_{\text{material}} \right] \end{aligned}$$

n= vettore normale associato al vertice

l= vettore dal vertice alla sorgente di luce (vertex to light)

h= halfway, orientato lungo la bisettrice tra **l** e **v** (vettore dal vertice all'osservatore, vertex to eye)

Normali

Se si vogliono utilizzare sorgenti di luce, è necessario indicare con quale angolo la luce incide sulla superficie illuminata, cioè bisogna specificare le normali per ogni vertice del poligono. Per individuare la normale è sufficiente considerare due vettori di un poligono e farne il prodotto vettoriale, quindi normalizzarlo in modo che abbia modulo unitario. Nella maggior parte dei casi bisogna predisporre alcune funzioni per il calcolo delle normali.

In aggiunta alle impostazioni di illuminazione e al calcolo delle normali occorre specificare il modello di shading (ombreggiature) delle superfici. Esistono due modalità di shading:

1) FLAT shading (si abilita con **glShadeModel(GL_FLAT)**)

A tutti i vertici della primitiva viene assegnato lo stesso colore selezionato da uno dei suoi vertici di cui si conosce la normale.

2) SMOOTH shading (si abilita con **glShadeModel(GL_SMOOTH)**)

Utilizza la modalità di shading di tipo interpolato. OpenGL calcola in modo esatto il colore dei vertici del poligono utilizzando le informazioni sulle normali.

I colori dei punti interni al poligono sono calcolati mediante interpolazione.

Normali

Dopo aver calcolato le componenti del vettore normale di un vertice si utilizza la seguente funzione per impostare il vettore:

```
void glNormal3fv(const GLfloat *v)
```

Di seguito vediamo le funzioni che possono essere utilizzate per il calcolo delle normali. Per definizione il vettore normale ad un poligono è ottenibile come prodotto vettoriale di due vettori collocati sulla superficie del poligono. Quindi è sufficiente conoscere le coordinate di 3 vertici del poligono.

```
void glGetNormalVector(const GLTVector3 vP1, const GLTVector3 vP2, const GLTVector3 vP3,  
    GLTVector3 vNormal)  {  
  
    GLTVector3 vV1, vV2;  
  
    gltSubtractVectors(vP2, vP1, vV1);  
  
    gltSubtractVectors(vP3, vP1, vV2);  
  
    gltVectorCrossProduct(vV1, vV2, vNormal);  
  
    gltNormalizeVector(vNormal);  
  
}
```

Normali (2)

```
void gltSubtractVectors(const GLTVector3 vFirst, const GLTVector3 vSecond, GLTVector3 vResult) {  
    vResult[0] = vFirst[0] - vSecond[0];  
    vResult[1] = vFirst[1] - vSecond[1];  
    vResult[2] = vFirst[2] - vSecond[2];  
}
```

```
void gltVectorCrossProduct(const GLTVector3 vU, const GLTVector3 vV, GLTVector3 vResult){  
    vResult[0] = vU[1]*vV[2] - vV[1]*vU[2];  
    vResult[1] = -vU[0]*vV[2] + vV[0]*vU[2];  
    vResult[2] = vU[0]*vV[1] - vV[0]*vU[1];  
}
```

Normali (3)

```
void gltNormalizeVector(GLTVector3 vNormal) {  
    GLfloat fLength = 1.0f / gltGetVectorLength(vNormal);  
    gltScaleVector(vNormal, fLength);  
}  
  
void gltScaleVector(GLTVector3 vVector, const GLfloat fScale) {  
    vVector[0] *= fScale; vVector[1] *= fScale; vVector[2] *= fScale;  
}  
  
GLfloat gltGetVectorLength(const GLTVector3 vVector) {  
    return (GLfloat)sqrt((vVector[0]*vVector[0]) +  
    (vVector[1]*vVector[1]) + (vVector[2]*vVector[2]) );  
}
```

Normali (4)

Anche le normali, come le coordinate dei vertici, subiscono le trasformazioni geometriche.

Le specifiche di OpenGL richiedono che le normali ai poligoni abbiano sempre norma unitaria. Se i vettori normali non vengono normalizzati correttamente si ha un comportamento di illuminazione anomalo.

Problema: le operazioni di scala (`glScalef`) alterano la norma dei vettori normali.

Quindi quando si utilizzano operazioni di scala occorre rinormalizzare i vettori normali.

L'istruzione `glEnable(GL_NORMALIZE)`; abilita la normalizzazione automatica delle normali e va chiamata una sola volta nel programma in fase di inizializzazione.

La funzione consente di passare a `glNormal` vettori di norma non necessariamente unitaria ma comporta un costo computazionale.

In alternativa è necessario normalizzare “manualmente” i vettori normali scalati con la funzione `glScalef`.

Two-sided lighting

E' possibile impostare OpenGL affinché il vettore normale delle facce back sia automaticamente invertito. Pertanto le facce back visibili all'osservatore vengono correttamente illuminate (per esempio le facce interne di una sfera non chiusa, con dei buchi).

Per l'illuminazione delle facce back si devono impostare i parametri dei materiali con l'opzione `GL_BACK` nella funzione **glMaterialfv**.

Di default è disabilitato (one-sided lighting).

Si abilita con la seguente riga di codice:

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

Le facce back devono essere disegnate disabilitando il back face culling.

Esempi

Esempio: il Jet

Negli esempi che seguono disegneremo un “Jet”, prima senza abilitare alcuna luce, quindi solo con la luce ambientale, infine con la luce diffusa e speculare.

Esempio 1: Jet senza luci

```
#include<GL/glut.h>

#include<GL/glu.h>

#include<GL/gl.h>

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

void RenderScene(void)  {

    glClear(GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT);

    glPushMatrix();

    glRotatef(xRot, 1.0f, 0.0f, 0.0f);

    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    // Nose Cone //////////////////////////////////////

    // White

    glColor3ub(255, 255, 255);

    glBegin(GL_TRIANGLES);

        glVertex3f(0.0f, 0.0f, 60.0f);

        glVertex3f(-15.0f, 0.0f, 30.0f);

        glVertex3f(15.0f,0.0f,30.0f);

    // Black

    glColor3ub(0,0,0);

    glVertex3f(15.0f,0.0f,30.0f);

    glVertex3f(0.0f, 15.0f, 30.0f);

    glVertex3f(0.0f, 0.0f, 60.0f);
```

Esempio 1: Jet senza luci (2)

// Red

```
glColor3ub(255,0,0);
```

```
glVertex3f(0.0f, 0.0f, 60.0f);
```

```
glVertex3f(0.0f, 15.0f, 30.0f);
```

```
glVertex3f(-15.0f,0.0f,30.0f);
```

// Body of the Plane //////////////////////////////////

// Green

```
glColor3ub(0,255,0);
```

```
glVertex3f(-15.0f,0.0f,30.0f);
```

```
glVertex3f(0.0f, 15.0f, 30.0f);
```

```
glVertex3f(0.0f, 0.0f, -56.0f);
```

```
glVertex3f(0.0f, 15.0f, 30.0f);
```

```
glVertex3f(15.0f,0.0f,30.0f);
```

```
glColor3ub(0, 255, 255);
```

```
glVertex3f(15.0f,0.0f,30.0f);
```

```
glVertex3f(-15.0f, 0.0f, 30.0f);
```

```
glVertex3f(0.0f, 0.0f, -56.0f);
```

Esempio 1: Jet senza luci (3)

/ Left wing

// Large triangle for bottom of wing

```
glColor3ub(128,128,128);
```

```
glVertex3f(0.0f,2.0f,27.0f);
```

```
glVertex3f(-60.0f, 2.0f, -8.0f);
```

```
glVertex3f(60.0f, 2.0f, -8.0f);
```

```
glColor3ub(64,64,64);
```

```
glVertex3f(60.0f, 2.0f, -8.0f);
```

```
glVertex3f(0.0f, 7.0f, -8.0f);
```

```
glVertex3f(0.0f,2.0f,27.0f);
```

```
glColor3ub(192,192,192);
```

```
glVertex3f(60.0f, 2.0f, -8.0f);
```

```
glVertex3f(-60.0f, 2.0f, -8.0f);
```

```
glVertex3f(0.0f,7.0f,-8.0f);
```

// Other wing top section

```
glColor3ub(64,64,64);
```

```
glVertex3f(0.0f,2.0f,27.0f);
```

```
glVertex3f(0.0f, 7.0f, -8.0f);
```

```
glVertex3f(-60.0f, 2.0f, -8.0f);
```

Esempio 1: Jet senza luci (4)

// Tail section////////////////////////////////////

// Bottom of back fin

glColor3ub(255,128,255);

glVertex3f(-30.0f, -0.50f, -57.0f);

glVertex3f(30.0f, -0.50f, -57.0f);

glVertex3f(0.0f,-0.50f,-40.0f);

// top of left side

glColor3ub(255,128,0);

glVertex3f(0.0f,-0.5f,-40.0f);

glVertex3f(30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

// top of right side

glColor3ub(255,128,0);

glVertex3f(0.0f, 4.0f, -57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f,-0.5f,-40.0f);

// back of bottom of tail

glColor3ub(255,255,255);

glVertex3f(30.0f,-0.5f,-57.0f);

glVertex3f(-30.0f, -0.5f, -57.0f);

glVertex3f(0.0f, 4.0f, -57.0f);

Esempio 1: Jet senza luci (5)

// Top of Tail section left

```
glColor3ub(255,0,0);
```

```
glVertex3f(0.0f,0.5f,-40.0f);
```

```
glVertex3f(3.0f, 0.5f, -57.0f);
```

```
glVertex3f(0.0f, 25.0f, -65.0f);
```

```
glColor3ub(255,0,0);
```

```
glVertex3f(0.0f, 25.0f, -65.0f);
```

```
glVertex3f(-3.0f, 0.5f, -57.0f);
```

```
glVertex3f(0.0f,0.5f,-40.0f);
```

// Back of horizontal section

```
glColor3ub(128,128,128);
```

```
glVertex3f(3.0f,0.5f,-57.0f);
```

```
glVertex3f(-3.0f, 0.5f, -57.0f);
```

```
glVertex3f(0.0f, 25.0f, -65.0f);
```

```
glEnd(); // Of Jet
```

```
glPopMatrix();
```

```
glutSwapBuffers();
```

```
}
```

Esempio 1: Jet senza luci (6)

```
void SetupRC() {  
    glEnable(GL_DEPTH_TEST);  
    glEnable(GL_CULL_FACE);  
    glFrontFace(GL_CCW);  
    glClearColor(0.0f, 0.0f, 0.5f, 1.0f);  
}  
  
void SpecialKeys(int key, int x, int y) {  
    if(key == GLUT_KEY_UP)  
        xRot -= 5.0f;  
    if(key == GLUT_KEY_DOWN)  
        xRot += 5.0f;  
  
    if(key == GLUT_KEY_LEFT)  
        yRot -= 5.0f;  
    if(key == GLUT_KEY_RIGHT)  
        yRot += 5.0f;  
  
    if(xRot > 356.0f)  
        xRot = 0.0f;  
    if(xRot < -1.0f)  
        xRot = 355.0f;  
    if(yRot > 356.0f)  
        yRot = 0.0f;
```

Esempio 1: Jet senza luci (7)

```
if(yRot < -1.0f)
```

```
    yRot = 355.0f;
```

```
    glutPostRedisplay();
```

```
}
```

```
void ChangeSize(int w, int h) {
```

```
    GLfloat nRange = 80.0f;
```

```
    if(h == 0)    h = 1;
```

```
    glViewport(0, 0, w, h);
```

```
    glMatrixMode(GL_PROJECTION);
```

```
    glLoadIdentity();
```

```
    if (w <= h)
```

```
        glOrtho (-nRange, nRange, -nRange*h/w,  
                nRange*h/w, -nRange, nRange);
```

```
    else
```

```
        glOrtho (-nRange*w/h, nRange*w/h,  
                -nRange, nRange, -nRange, nRange);
```

```
    glMatrixMode(GL_MODELVIEW);
```

```
    glLoadIdentity();
```

```
}
```

Esempio 1: Jet senza luci (8)

```
int main(int argc, char* argv[]) {  
    glutInit(&argc, argv);  
  
    glutInitDisplayMode(GLUT_DOUBLE |  
        GLUT_RGB | GLUT_DEPTH);  
  
    glutInitWindowSize(800,600);  
  
    glutCreateWindow("Jet");  
  
    glutReshapeFunc(ChangeSize);  
  
    glutSpecialFunc(SpecialKeys);  
  
    glutDisplayFunc(RenderScene);  
  
    SetupRC();  
  
    glutMainLoop();  
  
    return 0; }
```

Il risultato è un oggetto costituito da triangoli che assomiglia ad un jet, ma l'effetto di tridimensionalità è dato dalla diversità dei colori: infatti se commentiamo tutti i glColor dopo il primo, l'oggetto (che è in 3D) sembra una sagoma piatta.

Esempio 2 :Jet + luce ambiente

```
#include<GL/glut.h>
```

```
#include<GL/glu.h>
```

```
#include<GL/gl.h>
```

```
// Rotation amounts
```

```
static GLfloat xRot = 0.0f;
```

```
static GLfloat yRot = 0.0f;
```

```
void RenderScene() come nell'esempio  
precedente
```

```
void SetupRC() {
```

```
    GLfloat ambientLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
```

```
    glEnable(GL_DEPTH_TEST);
```

```
    glEnable(GL_CULL_FACE);
```

```
    glFrontFace(GL_CCW);
```

```
    glEnable(GL_LIGHTING);
```

```
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  
        ambientLight);
```

```
    glEnable(GL_COLOR_MATERIAL);
```

```
    glColorMaterial(GL_FRONT,  
        GL_AMBIENT_AND_DIFFUSE);
```

```
    glClearColor(0.0f, 0.0f, 0.5f, 1.0f);
```

```
}
```

Esempio 2 :Jet + luce ambiente (2)

`void SpecialKeys(int key, int x, int y) e`

`void ChangeSize(int w, int h) e`

`int main(int argc, char* argv[])`

come nell'esempio precedente

Si nota che dall'esempio precedente non cambia molto, ma se nella funzione `SetupRC` cambiamo l' intensità della luce d'ambiente (ad esempio 0.1 0.1 0.1) vediamo la scena diventare più buia

Cosa accade ai colori del jet se le componenti RGB della luce sono diverse ?

Nell'esempio seguente è mostrato l'effetto della luce diffusa che colpisce il jet, per questo la funzione `RenderScene` andrà modificata specificando un solo colore (grigio) per tutti i triangoli, con `glColor3ub(128, 128, 128);`

Vengono inoltre utilizzate le funzioni per il calcolo delle normali viste in precedenza, quindi andranno copiate all'interno del codice sorgente

Esempio 3: Jet +luce diffusa

```
#include<GL/glut.h>

#include<GL/glu.h>

#include<GL/gl.h>

#include <math.h>

typedef GLfloat GLTVector3[3];

// per il calcolo delle normali...

static GLfloat xRot = 0.0f;

static GLfloat yRot = 0.0f;

void RenderScene(void){

    GLTVector3 vNormal;

    glClear(GL_COLOR_BUFFER_BIT |
           GL_DEPTH_BUFFER_BIT);

    glPushMatrix();

    glRotatef(xRot, 1.0f, 0.0f, 0.0f);

    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    // Nose Cone - Points straight down

    glColor3ub(128, 128, 128);

    glBegin(GL_TRIANGLES);

    glNormal3f(0.0f, -1.0f, 0.0f);

    glNormal3f(0.0f, -1.0f, 0.0f);

    glVertex3f(0.0f, 0.0f, 60.0f);

    glVertex3f(-15.0f, 0.0f, 30.0f);

    glVertex3f(15.0f,0.0f,30.0f);
```

Esempio 3:Jet+luce diffusa (2)

<pre>{ GLTVector3 vPoints[3] = { { 15.0f, 0.0f, 30.0f},{ 0.0f, 15.0f, 30.0f}, 0.0f, 0.0f, 60.0f}}; gltGetNormalVector(vPoints[0], vPoints[1], vPoints[2], vNormal); glNormal3fv(vNormal); glVertex3fv(vPoints[0]); glVertex3fv(vPoints[1]); glVertex3fv(vPoints[2]); }</pre>	<pre>{ GLTVector3 vPoints[3] = { { 0.0f, 0.0f, 60.0f },{ 0.0f, 15.0f, 30.0f }, { -15.0f, 0.0f, 30.0f }}; gltGetNormalVector(vPoints[0], vPoints[1], vPoints[2], vNormal); glNormal3fv(vNormal); glVertex3fv(vPoints[0]); glVertex3fv(vPoints[1]); glVertex3fv(vPoints[2]); }</pre>
---	---

Esempio 3:Jet+luce diffusa (2)

```
// Body of the Plane ////////////////////////////////////
{
GLTVector3 vPoints[3] = {{ -15.0f, 0.0f, 30.0f },
{ 0.0f, 15.0f, 30.0f },{ 0.0f, 0.0f, -56.0f }};

gltGetNormalVector(vPoints[0], vPoints[1],
    vPoints[2], vNormal);

glNormal3fv(vNormal);

glVertex3fv(vPoints[0]);

glVertex3fv(vPoints[1]);

glVertex3fv(vPoints[2]);
}

{
GLTVector3 vPoints[3] = {{ 0.0f, 0.0f, -56.0f },
{ 0.0f, 15.0f, 30.0f },{ 15.0f,0.0f,30.0f }};

gltGetNormalVector(vPoints[0], vPoints[1],
    vPoints[2], vNormal);

glNormal3fv(vNormal);

glVertex3fv(vPoints[0]);

glVertex3fv(vPoints[1]);

glVertex3fv(vPoints[2]);
}
```

Esempio 3:Jet+luce diffusa (2)

```
glNormal3f(0.0f, -1.0f, 0.0f);
```

```
glVertex3f(15.0f,0.0f,30.0f);
```

```
glVertex3f(-15.0f, 0.0f, 30.0f);
```

```
glVertex3f(0.0f, 0.0f, -56.0f);
```

```
// Left wing, large triangle for bottom of wing
```

```
{
```

```
GLTVector3 vPoints[3] = {{ 0.0f,2.0f,27.0f },
```

```
{ -60.0f, 2.0f, -8.0f },{ 60.0f, 2.0f, -8.0f }};
```

```
gltGetNormalVector(vPoints[0], vPoints[1],  
vPoints[2], vNormal);
```

```
glNormal3fv(vNormal);
```

```
glVertex3fv(vPoints[0]);
```

```
glVertex3fv(vPoints[1]);
```

```
glVertex3fv(vPoints[2]);
```

```
}
```

```
{
```

```
GLTVector3 vPoints[3] = {{ 60.0f, 2.0f, -8.0f}  
,{0.0f, 7.0f, -8.0f},{0.0f,2.0f,27.0f }};
```

```
gltGetNormalVector(vPoints[0], vPoints[1],  
vPoints[2], vNormal);
```

```
glNormal3fv(vNormal);
```

```
glVertex3fv(vPoints[0]);
```

```
glVertex3fv(vPoints[1]);
```

```
glVertex3fv(vPoints[2]);
```

```
}
```

Esempio 3:Jet+luce diffusa (3)

```
{  
    GLTVector3 vPoints[3] = {{60.0f, 2.0f, -8.0f},  
    {-60.0f, 2.0f, -8.0f},{0.0f,7.0f,-8.0f } };  
  
    glGetNormalVector(vPoints[0], vPoints[1],  
        vPoints[2], vNormal);  
  
    glNormal3fv(vNormal);  
  
    glVertex3fv(vPoints[0]);  
  
    glVertex3fv(vPoints[1]);  
  
    glVertex3fv(vPoints[2]);  
}
```

```
{  
    GLTVector3 vPoints[3] = {{0.0f,2.0f,27.0f},  
    {0.0f, 7.0f, -8.0f},{-60.0f, 2.0f, -8.0f} };  
  
    glGetNormalVector(vPoints[0], vPoints[1],  
        vPoints[2], vNormal);  
  
    glNormal3fv(vNormal);  
  
    glVertex3fv(vPoints[0]);  
  
    glVertex3fv(vPoints[1]);  
  
    glVertex3fv(vPoints[2]);  
}
```

Esempio 3:Jet+luce diffusa (4)

```
// Tail section bttom of back fin
```

```
glNormal3f(0.0f, -1.0f, 0.0f);
```

```
glVertex3f(-30.0f, -0.50f, -57.0f);
```

```
glVertex3f(30.0f, -0.50f, -57.0f);
```

```
glVertex3f(0.0f,-0.50f,-40.0f);
```

```
{
```

```
GLTVector3 vPoints[3] = { { 0.0f,-0.5f,-40.0f },
```

```
{30.0f, -0.5f, -57.0f},{0.0f, 4.0f, -57.0f } };
```

```
glGetNormalVector(vPoints[0], vPoints[1],  
vPoints[2], vNormal);
```

```
glNormal3fv(vNormal);
```

```
glVertex3fv(vPoints[0]);
```

```
glVertex3fv(vPoints[1]);
```

```
glVertex3fv(vPoints[2]);
```

```
}
```

```
{
```

```
GLTVector3 vPoints[3] = { { 0.0f, 4.0f, -57.0f }, {  
-30.0f, -0.5f, -57.0f }, { 0.0f,-0.5f,-40.0f } };
```

```
glGetNormalVector(vPoints[0], vPoints[1],  
vPoints[2], vNormal);
```

```
glNormal3fv(vNormal);
```

```
glVertex3fv(vPoints[0]);
```

```
glVertex3fv(vPoints[1]);
```

```
glVertex3fv(vPoints[2]);
```

```
}
```

Esempio 3:Jet+luce diffusa (5)

```
{  
    GLTVector3 vPoints[3] = {{ 30.0f,-  
        0.5f,-57.0f },  
    { -30.0f, -0.5f, -57.0f },  
    { 0.0f, 4.0f, -57.0f }};  
    glGetNormalVector(vPoints[0], vPoints[1],  
        vPoints[2], vNormal);  
    glNormal3fv(vNormal);  
    glVertex3fv(vPoints[0]);  
    glVertex3fv(vPoints[1]);  
    glVertex3fv(vPoints[2]);  
}
```

```
{  
    GLTVector3 vPoints[3] = {{ 0.0f,0.5f,-40.0f },  
    { 3.0f, 0.5f, -57.0f },{ 0.0f, 25.0f, -65.0f }};  
  
    glGetNormalVector(vPoints[0], vPoints[1],  
        vPoints[2], vNormal);  
    glNormal3fv(vNormal);  
    glVertex3fv(vPoints[0]);  
    glVertex3fv(vPoints[1]);  
    glVertex3fv(vPoints[2]);  
}
```

Esempio 3:Jet+luce diffusa (6)

```
{  
    GLTVector3 vPoints[3] = {{ 0.0f,  
        25.0f, -65.0f },  
    { -3.0f, 0.5f, -57.0f},  
    { 0.0f,0.5f,-40.0f } };  
  
    gltGetNormalVector(vPoints[0],  
        vPoints[1], vPoints[2], vNormal);  
  
    glNormal3fv(vNormal);  
  
    glVertex3fv(vPoints[0]);  
  
    glVertex3fv(vPoints[1]);  
  
    glVertex3fv(vPoints[2]);  
}
```

```
{  
    GLTVector3 vPoints[3] = {{ 3.0f,0.5f,-57.0f },  
    { -3.0f, 0.5f, -57.0f },{ 0.0f, 25.0f, -65.0f } };  
  
    gltGetNormalVector(vPoints[0], vPoints[1],  
        vPoints[2], vNormal);  
  
    glNormal3fv(vNormal);  
  
    glVertex3fv(vPoints[0]);  
  
    glVertex3fv(vPoints[1]);  
  
    glVertex3fv(vPoints[2]);  
}  
  
    glEnd();  
  
    glPopMatrix();  
  
    glutSwapBuffers(); }
```

Esempio 3:Jet+luce diffusa (7)

```
void SetupRC() {  
    GLfloat ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };  
    GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };  
    glEnable(GL_DEPTH_TEST);  
    glFrontFace(GL_CCW);  
    glEnable(GL_CULL_FACE);  
    glEnable(GL_LIGHTING);  
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);  
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);  
    glEnable(GL_LIGHT0);  
    glEnable(GL_COLOR_MATERIAL);  
    glColorMaterial(GL_FRONT,  
                   GL_AMBIENT_AND_DIFFUSE);  
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f );  
}
```

Esempio 4:Jet+luce speculare

Per abilitare la luce speculare (riflessi) basta modificare la funzione SetupRC come segue:

```
void SetupRC() {  
    GLfloat ambientLight[] = { 0.3f, 0.3f, 0.3f, 1.0f };  
    GLfloat diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };  
    GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
    GLfloat specref[] = { 1.0f, 1.0f, 1.0f, 1.0f };  
    glEnable(GL_DEPTH_TEST);  
    glFrontFace(GL_CCW);  
    glEnable(GL_CULL_FACE);  
    glEnable(GL_LIGHTING);
```

```
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLight);  
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuseLight);  
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);  
    glEnable(GL_LIGHT0);  
    glEnable(GL_COLOR_MATERIAL);  
    glColorMaterial(GL_FRONT,  
        GL_AMBIENT_AND_DIFFUSE);  
    glMaterialfv(GL_FRONT, GL_SPECULAR, specref);  
    glMateriali(GL_FRONT, GL_SHININESS, 128);  
    glClearColor(0.0f, 0.0f, 1.0f, 1.0f );  
}
```

Esempio 5 :light spot

In questo esempio si mostra come utilizzare una luce posizionale per ottenere un effetto simile a quello di un faro puntato su un oggetto.

Per ottenere uno “spotlight” da una luce posizionale si chiama

```
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 60.0f)
```

con cui si specifica il semiangolo di apertura del faro (valore ammessi compresi tra 0 e 90).

Si utilizza inoltre la seguente funzione per definire la direzione del fascio rispetto alla sorgente

```
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir)
```

Le luci di questo tipo illuminano solo gli oggetti nel loro cono.

In questo esempio viene inoltre mostrata la differenza tra un modello di shading flat e uno smooth.

Esempio 5 :light spot (2)

```
#include<GL/glut.h>
```

```
#include<GL/glu.h>
```

```
#include<GL/gl.h>
```

```
static GLfloat xRot = 0.0f;
```

```
static GLfloat yRot = 0.0f;
```

```
GLfloat lightPos[] = { 0.0f, 0.0f, 75.0f, 1.0f };
```

```
GLfloat specular[] = { 1.0f, 1.0f, 1.0f, 1.0f};
```

```
GLfloat specref[] = { 1.0f, 1.0f, 1.0f, 1.0f };
```

```
GLfloat ambientLight[] = { 0.5f, 0.5f, 0.5f, 1.0f};
```

```
GLfloat spotDir[] = { 0.0f, 0.0f, -1.0f };
```

```
#define MODE_FLAT 1
```

```
#define MODE_SMOOTH 2
```

```
#define MODE_VERYLOW 3
```

```
#define MODE_MEDIUM 4
```

```
#define MODE_VERYHIGH 5
```

```
int iShade = MODE_FLAT;
```

```
int iTess = MODE_VERYLOW;
```

```
void ProcessMenu(int value) {
```

```
    switch(value) {
```

```
        case 1:
```

```
            iShade = MODE_FLAT;
```

```
            break;
```

Esempio 5 :light spot (3)

```
case 2:
    iShade = MODE_SMOOTH;
    break;
case 3:
    iTess = MODE_VERYLOW;
    break;
case 4:
    iTess = MODE_MEDIUM;
    break;
case 5:
    default:
        iTess = MODE_VERYHIGH;
        break;
}

glutPostRedisplay();
}

void RenderScene(void) {
    if(iShade == MODE_FLAT)
        glShadeModel(GL_FLAT);
    else    glShadeModel(GL_SMOOTH);

    glClear(GL_COLOR_BUFFER_BIT |
            GL_DEPTH_BUFFER_BIT);

    glPushMatrix();

    glRotatef(yRot, 0.0f, 1.0f, 0.0f);
    glRotatef(xRot, 1.0f, 0.0f, 0.0f);
```

Esempio 5 :light spot (4)

```
glLightfv(GL_LIGHT0, GL_POSITION, lightPos);  
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spotDir);  
glColor3ub(255, 0, 0);  
glTranslatef(lightPos[0], lightPos[1], lightPos[2]);  
glutSolidCone(4.0f, 6.0f, 15, 15);  
glDisable(GL_LIGHTING);  
glColor3ub(255, 255, 0);  
glutSolidSphere(3.0f, 15, 15);  
glEnable(GL_LIGHTING);  
  
glPopMatrix();  
glColor3ub(0, 0, 255);  
if(iTess == MODE_VERYLOW)  
    glutSolidSphere(30.0f, 7, 7);  
else  
    if(iTess == MODE_MEDIUM)  
        glutSolidSphere(30.0f, 15, 15);  
    else  
        glutSolidSphere(30.0f, 50, 50);  
glutSwapBuffers();  
}
```

Esempio 5 :light spot (5)

```
void SetupRC() {  
  
    glEnable(GL_DEPTH_TEST);  
  
    glFrontFace(GL_CCW);  
  
    glEnable(GL_CULL_FACE);  
  
    glEnable(GL_LIGHTING);  
  
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT,  
        ambientLight);  
  
    glLightfv(GL_LIGHT0, GL_DIFFUSE, ambientLight);  
  
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular);  
  
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos);  
  
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 50.0f);  
  
    glEnable(GL_LIGHT0);
```

```
    glEnable(GL_COLOR_MATERIAL);  
  
    glColorMaterial(GL_FRONT,  
        GL_AMBIENT_AND_DIFFUSE);  
  
    glMaterialfv(GL_FRONT, GL_SPECULAR, specref);  
  
    glMateriali(GL_FRONT, GL_SHININESS, 128);  
  
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f );  
  
}
```

`void SpecialKeys(int key, int x, int y)` come negli esempi precedenti

```
void ChangeSize(int w, int h) {  
  
    GLfloat fAspect;  
  
    if(h == 0)    h = 1;
```

Esempio 5 :light spot (6)

```
glViewport(0, 0, w, h);  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
fAspect = (GLfloat) w / (GLfloat) h;  
gluPerspective(35.0f, fAspect, 1.0f, 500.0f);  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
glTranslatef(0.0f, 0.0f, -250.0f);  
}
```

```
int main(int argc, char* argv[]) {  
    int nMenu;
```

```
    glutInitDisplayMode(GLUT_DOUBLE |  
        GLUT_RGB | GLUT_DEPTH);  
    glutInitWindowSize(800, 600);  
    glutCreateWindow("Spot Light");  
    nMenu = glutCreateMenu(ProcessMenu);  
    glutAddMenuEntry("Flat Shading",1);  
    glutAddMenuEntry("Smooth Shading",2);  
    glutAddMenuEntry("VL Tess",3);  
    glutAddMenuEntry("MD Tess",4);  
    glutAddMenuEntry("VH Tess",5);  
    glutAttachMenu(GLUT_RIGHT_BUTTON);  
    // varie chiamate glut  
}
```

Esercizio

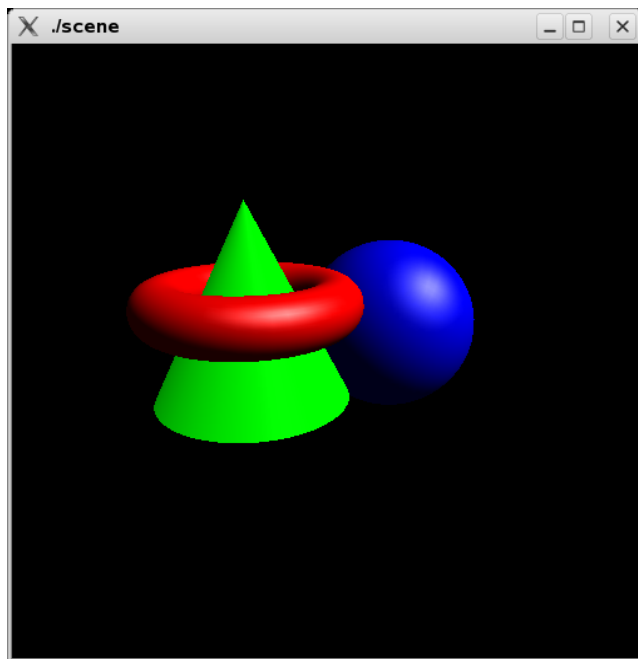
Scrivere un programma per visualizzare la scena mostrata in figura.

1) La scena contiene 3 oggetti: un cono ($r_{base}=1.0$, $altezza=2.0$), un toro ($r=0.275$, $R=0.85$) e una sfera ($r=1.0$). Il toro e' coassiale al cono. La sfera e' traslata lungo l'asse x di $+0.75$, il cono e il toro sono traslati lungo l'asse x di -0.75

2) E' presente una luce ambientale ($\{ 0.1f, 0.1f, 0.1f, 1.0f \}$)

3) E' presente una luce direzionale con componenti ambientale, diffusa e speculare. La luce e' direzionata in modo da illuminare tutti e tre gli oggetti (es: $\{ 1.0, 1.0, 1.0, 0.0 \}$)

4) Gli oggetti vanno disegnati attivando lo smooth shading, il depth buffer, il double buffering e la modalit  di "color tracking"



5) Si utilizzano 2 tasti per ruotare il cono e il toro rispetto ad un asse parallelo all'asse x che interseca l'asse del cono

6) Si utilizzano 2 tasti per ruotare l'intera scena rispetto all'asse y mantenendo la luce direzionale solidale agli oggetti

7) Definire una matrice di proiezione di tipo prospettico

Alpha Blending

La modalità di blending, che si abilita con `glEnable(GL_BLEND)`, fa in modo che i nuovi valori di colore associati ad un pixel che vengono scritti nel color buffer siano combinati con quelli già presenti, secondo un'equazione predefinita.

Chiamiamo *colore destinazione* quello già presente nel color buffer, mentre chiamiamo *colore sorgente* quello che è prodotto da una serie di chiamate di rendering (`glVertex` ecc...). Per default l'equazione di blending è

$$C_f = C_s * S + C_d * D$$

in cui C_f è il colore calcolato finale, C_s il colore sorgente e C_d il colore destinazione, S e D sono i fattori alpha della sorgente e della destinazione. I fattori S e D si settano con `void glBlendFunc(GLenum  $S$ , GLenum  $D$ )`. Una combinazione molto comune di S e D è data da `glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)`

Esempio 6

```
#include<GL/glut.h>
```

```
#include<GL/glu.h>
```

```
#include<GL/gl.h>
```

```
GLfloat fLightPos[4] = { -100.0f, 100.0f, 50.0f, 1.0f };
```

```
GLfloat fLightPosMirror[4] = { -100.0f, -100.0f, 50.0f,  
    1.0f };
```

```
GLfloat fNoLight[] = { 0.0f, 0.0f, 0.0f, 0.0f };
```

```
GLfloat fLowLight[] = { 0.25f, 0.25f, 0.25f, 1.0f };
```

```
GLfloat fBrightLight[] = { 1.0f, 1.0f, 1.0f, 1.0f };
```

```
static GLfloat yRot = 0.0f;
```

```
void SetupRC() {
```

```
    glClearColor(fLowLight[0], fLowLight[1],  
        fLowLight[2], fLowLight[3]);
```

```
    glCullFace(GL_BACK);
```

```
    glEnable(GL_CULL_FACE);
```

```
    glEnable(GL_DEPTH_TEST);  
    glLightModelfv(GL_LIGHT_MODEL_AMB  
        IENT, fNoLight);
```

```
    glLightfv(GL_LIGHT0, GL_AMBIENT,  
        fLowLight);
```

```
    glLightfv(GL_LIGHT0, GL_DIFFUSE,  
        fBrightLight);
```

```
    glLightfv(GL_LIGHT0, GL_SPECULAR,  
        fBrightLight);
```

```
    glEnable(GL_LIGHTING);
```

Esempio 6 (2)

```
glEnable(GL_LIGHT0);
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT,
    GL_AMBIENT_AND_DIFFUSE);
glMateriali(GL_FRONT, GL_SHININESS, 128);
}

void DrawGround(void) {
    GLfloat fExtent = 20.0f;
    GLfloat fStep = 0.5f;
    GLfloat y = 0.0f;
    GLfloat fColor;

    GLfloat iStrip, iRun;
    GLint iBounce = 0;
    glShadeModel(GL_FLAT);

    for(iStrip = -fExtent; iStrip <= fExtent; iStrip += fStep){
        glBegin(GL_TRIANGLE_STRIP);

        for(iRun = fExtent; iRun >= -fExtent; iRun -= fStep) {
            if((iBounce % 2) == 0)
                fColor = 1.0f;
            else
                fColor = 0.0f;
            glColor4f(fColor, fColor, fColor, 0.5f);
            glVertex3f(iStrip, y, iRun);
```

Esempio 6 (3)

```
        glVertex3f(iStrip + fStep, y, iRun);
iBounce++;
    }
    glEnd();
}

glShadeModel(GL_SMOOTH);
}

void DrawWorld(void) {
    glColor3f(1.0f, 0.0f, 0.0f);
    glPushMatrix();

    glTranslatef(0.0f, 0.5f, -3.5f);

    glPushMatrix();

    glRotatef(-yRot * 2.0f, 0.0f, 1.0f, 0.0f);

    glTranslatef(1.0f, 0.0f, 0.0f);

    glutSolidSphere(0.1f, 17, 9);

    glPopMatrix();

    glColor3f(1.0f, 1.0f, 0.0f);

    glRotatef(yRot, 0.0f, 1.0f, 0.0f);

    glutSolidTorus(0.15,0.35,60,40);

    glPopMatrix();
}
```

Esempio 6 (4)

```
void RenderScene(void) {  
    glClear(GL_COLOR_BUFFER_BIT |  
            GL_DEPTH_BUFFER_BIT);  
    glPushMatrix();  
    glLightfv(GL_LIGHT0, GL_POSITION,  
              fLightPosMirror);  
    glPushMatrix();  
    glFrontFace(GL_CW);           // geometry is  
    mirrored, swap orientation  
    glScalef(1.0f, -1.0f, 1.0f);  
    DrawWorld();  
    glFrontFace(GL_CCW);  
    glPopMatrix();  
    glDisable(GL_LIGHTING);  
    glEnable(GL_BLEND);  
    glBlendFunc(GL_SRC_ALPHA,  
                GL_ONE_MINUS_SRC_ALPHA);  
    DrawGround();  
    glDisable(GL_BLEND);  
    glEnable(GL_LIGHTING);  
    glLightfv(GL_LIGHT0, GL_POSITION,  
              fLightPos);  
    DrawWorld();  
    glPopMatrix();  
    glutSwapBuffers();  
}
```

Esempio 6 (5)

```
void TimerFunction(int value) {  
    yRot += 1.0f;  
    glutPostRedisplay();  
    glutTimerFunc(1, TimerFunction, 1);  
}  
  
void ChangeSize(int w, int h) {  
    GLfloat fAspect;  
    if(h == 0)    h = 1;  
    glViewport(0, 0, w, h);  
    fAspect = (GLfloat)w / (GLfloat)h;  
    glMatrixMode(GL_PROJECTION);  
    glLoadIdentity();  
  
    gluPerspective(35.0f, fAspect, 1.0f, 50.0f);  
    glMatrixMode(GL_MODELVIEW);  
    glLoadIdentity();  
    glTranslatef(0.0f, -0.4f, 0.0f);  
}  
  
int main(int argc, char* argv[])  
{  
    ...  
    glutDisplayFunc(RenderScene);  
    glutTimerFunc(10, TimerFunction, 1);  
    ...  
}
```

Nebbia (1)

OpenGL possiede funzioni specifiche per simulare l'effetto della nebbia.

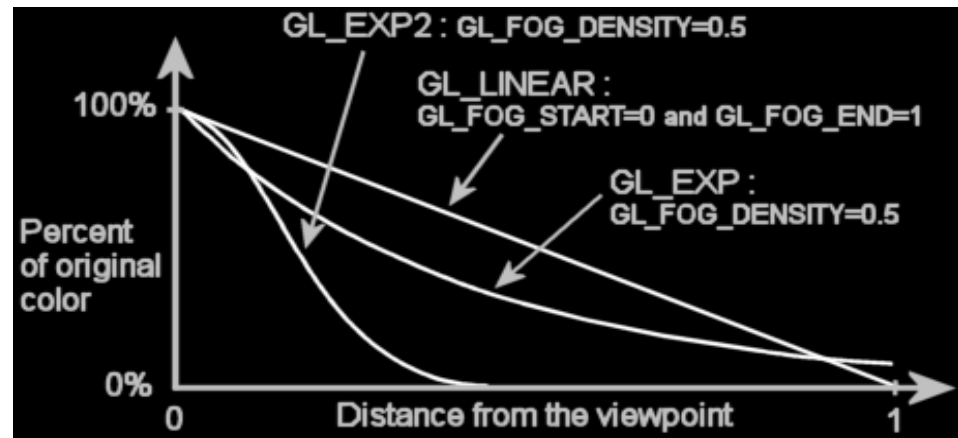
La nebbia deve avere un determinato colore. Es: `float fogColor[] = { 0.5f, 0.5f, 0.5f, 1.0f };`

Esistono 3 tipi di nebbia:

GL_EXP nebbia semplice.

GL_EXP2 decadimento esponenziale.

GL_LINEAR Versione più realistica.



Per impostare i parametri dell'effetto di nebbia si utilizzano le funzioni **glFog[fvi]**

GL_FOG_MODE Specifica il tipo di effetto utilizzato, come già visto.

GL_FOG_DENSITY > 0.0f (default 1.0f) Specifica la densità della nebbia.

GL_FOG_START Specifica la distanza vicina della nebbia. Non c' nebbia prima di questo punto.

GL_FOG_END Un qualsiasi float (default 1.0f). Specifica la distanza lontana della nebbia. Non c' nebbia oltre questo punto.

GL_FOG_COLOR imposta il colore della nebbia

Nebbia (2)

```
Esempio: GLfloat fog_color[] = { 1, 1, 1, 0 }; /* white */  
glFogfv(GL_FOG_COLOR, fog_color);
```

Un'altra funzione, non esclusiva della nebbia, è **glHint**. Questa funzione permette di specificare quanto sia importante il rapporto aspetto/velocità del programma

Il primo parametro accetta un certo numero di valori:

GL_FOG_HINT, GL_LINE_SMOOTH_HINT, GL_PERSPECTIVE_CORRECTION_HINT e GL_POINT_SMOOTH_HINT. Noi vedremo solo GL_FOG_HINT.

Il secondo parametro può essere GL_DONT_CARE, GL_FASTEST o GL_NICEST. Se vogliamo che l'effetto sia migliore possibile passeremo GL_NICEST, se più veloce possibile GL_FASTEST. Il valore di default GL_DONT_CARE migliora l'effetto quando ci sono risorse computazionali disponibili per farlo.

(ESEMPIO 7)

Texture Mapping (cenni)

Tecnica che consente di applicare delle immagini (“texture”) alle superfici da renderizzare.

Procedura:

- 1) Caricamento della texture (attraverso funzioni per caricare immagini nei vari formati es: bmp, jpg, gif, png ec...)
- 2) Mappare le coordinate della texture sul poligono (OPENGL lavora con immagini che hanno dimensioni che sono potenze di 2).

Funzioni principali:

glGenTextures crea un identificatore della texture

glBindTexture imposta la texture corrente

glTexImage2D definisce le proprietà principali della texture

glTexCoord specifica le coordinate della texture in 1,2 o 3 dimensioni.

(ESEMPIO8)