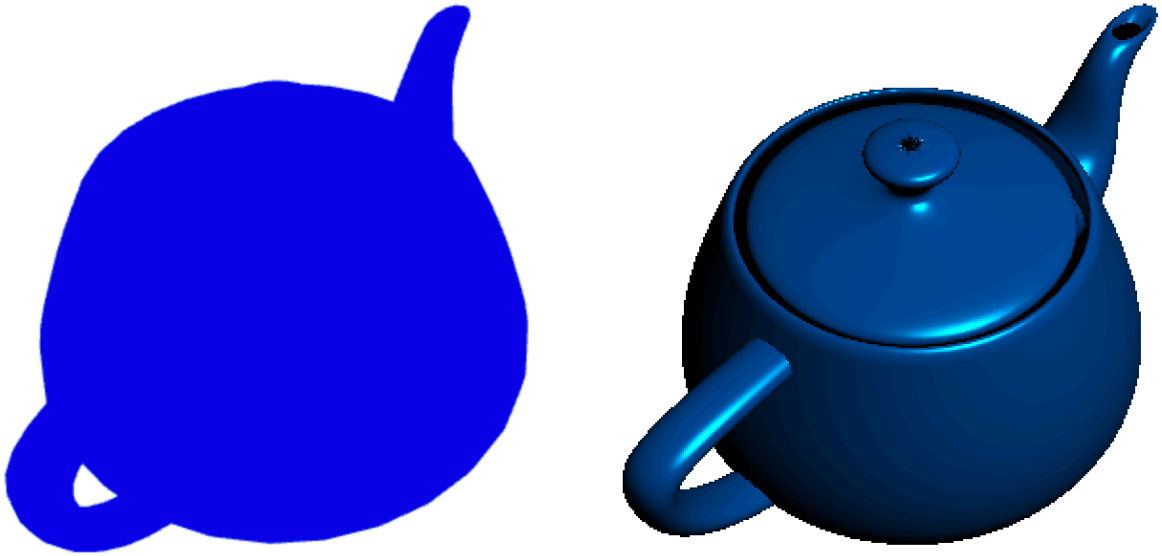
Modello di illuminazione di Phong



Illuminazione

Quando il nostro occhio vede un oggetto colorato, in realtà percepisce la luce che l'oggetto riflette perché viene illuminato da una sorgente luminosa.

La luce che colpisce un punto di un oggetto, viene in parte assorbita (e quindi trasformata in altre forme di energia come il calore), in parte riflessa e in parte può essere rifratta ovvero trasmessa all'interno della superficie dell'oggetto (rifrazione).

La quantità e il colore della luce riflessa dagli oggetti è ciò che ci consente di vederli come rossi, verdi o di altro colore.

Matematicamente, questo fenomeno fisico viene modellizzato dalla **Rendering equation**. Questa equazione per la sua complessità non può essere risolta analiticamente.

Illuminazione

I modelli di illuminazione che si assumono in computer graphics sono approssimazioni della rendering equation e possono essere di tipo locale o globale.

Un **modello di illuminazione locale** calcola il colore di un punto in base alla luce direttamente emessa dalle sorgenti luminose ed alle proprietà della superficie. In questo caso la luminosità di un punto non è influenzata da altri oggetti in scena.

Un **modello di illuminazione globale** calcola il colore di un punto in base alla luce direttamente emessa dalle sorgenti luminose e alla luce che raggiunge quel punto riflessa o trasmessa dalla superficie stessa o da altre superfici di oggetti presenti. L'illuminazione del punto è influenzata quindi anche dagli altri oggetti nella scena.

Componenti del Modello di Illuminazione di Phong

Il modello di Phong (1973) è un modello di illuminazione locale, cioè che simula la luminosità di una superficie colpita direttamente da una sorgente luminosa; in pratica:

luce finale = { (sorgente uniforme) illuminazione *ambiente* + (sorgente puntiforme) *riflessione* { *diffusa* + *speculare*

Componente *ambiente*

- Modella approssimativamente l'effetto di una luce uniforme che arriva da riflessioni secondarie
- Tutte le superfici, anche quelle in ombra, sono raggiunte da luce proveniente da tutte le direzioni

solo componente *ambiente*

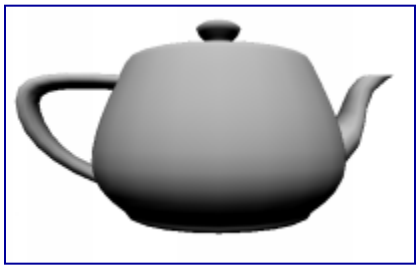
$$I_{da} = k_a I_a$$

$K_R, K_G, K_B \in [0,1]$
(definisce il colore dell'oggetto)

R, G, B
(luce bianca: 1,1,1)

Componente *riflessione diffusa*

- E' tipica in superfici come per es.:
  - materiali molto opachi (il contrario di lucidi)
  - certi tipi di legno
  - gesso
  - ….

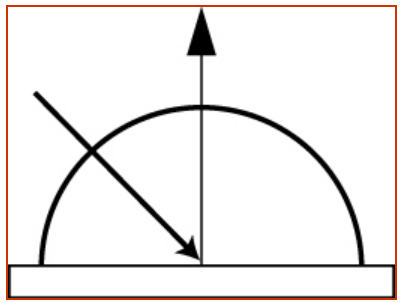


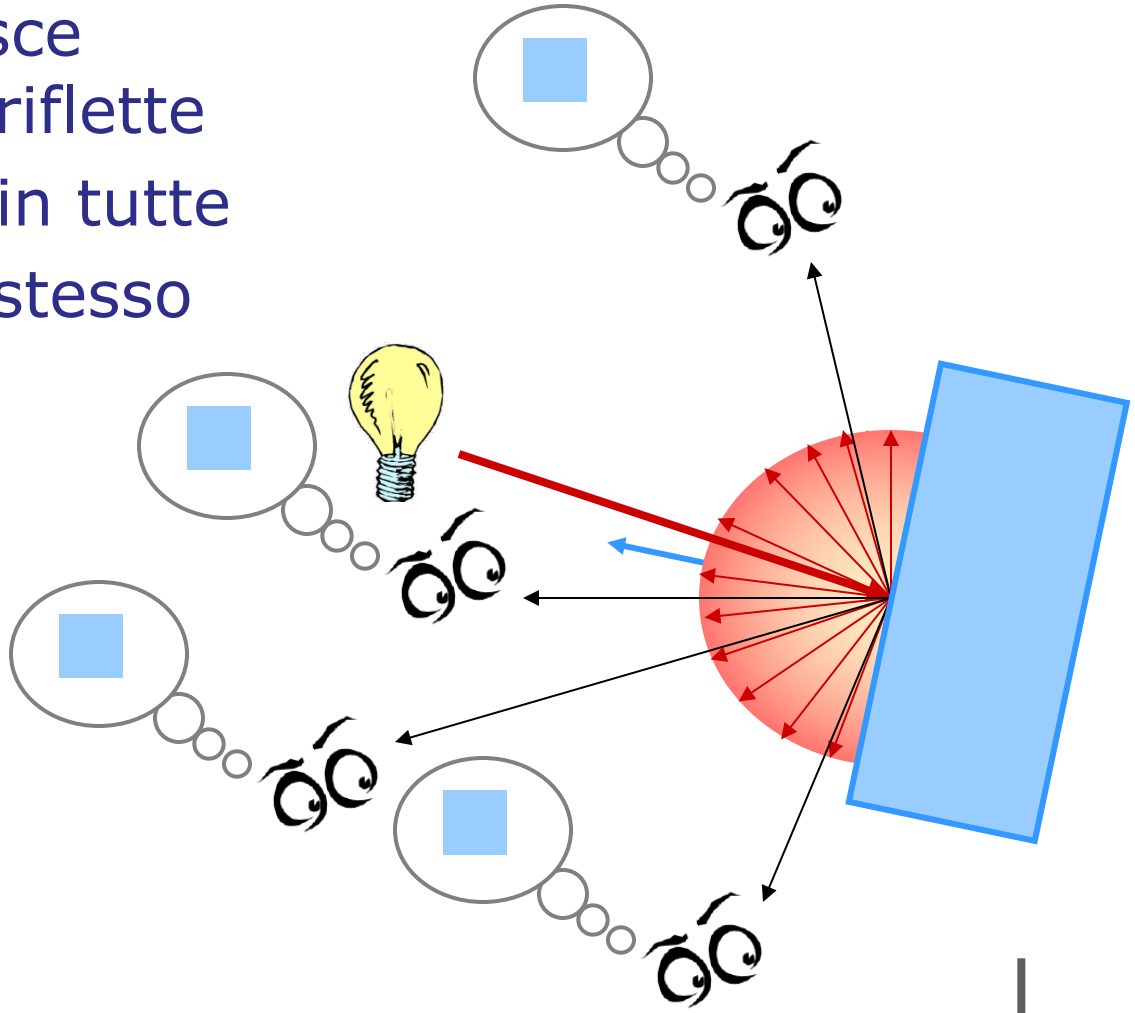
con componente *riflessione diffusa*

- Viene detta anche
  - Riflessione Lambertiana (da J.H.Lambert 1728-1777)

Componente *riflessione diffusa*

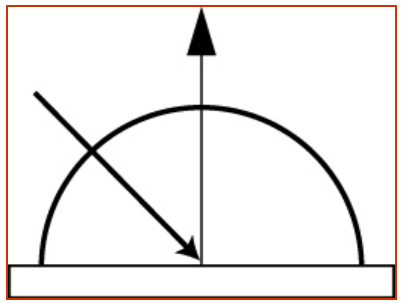
- La luce che colpisce una superficie si riflette (nel semispazio) in tutte le direzioni nello stesso modo

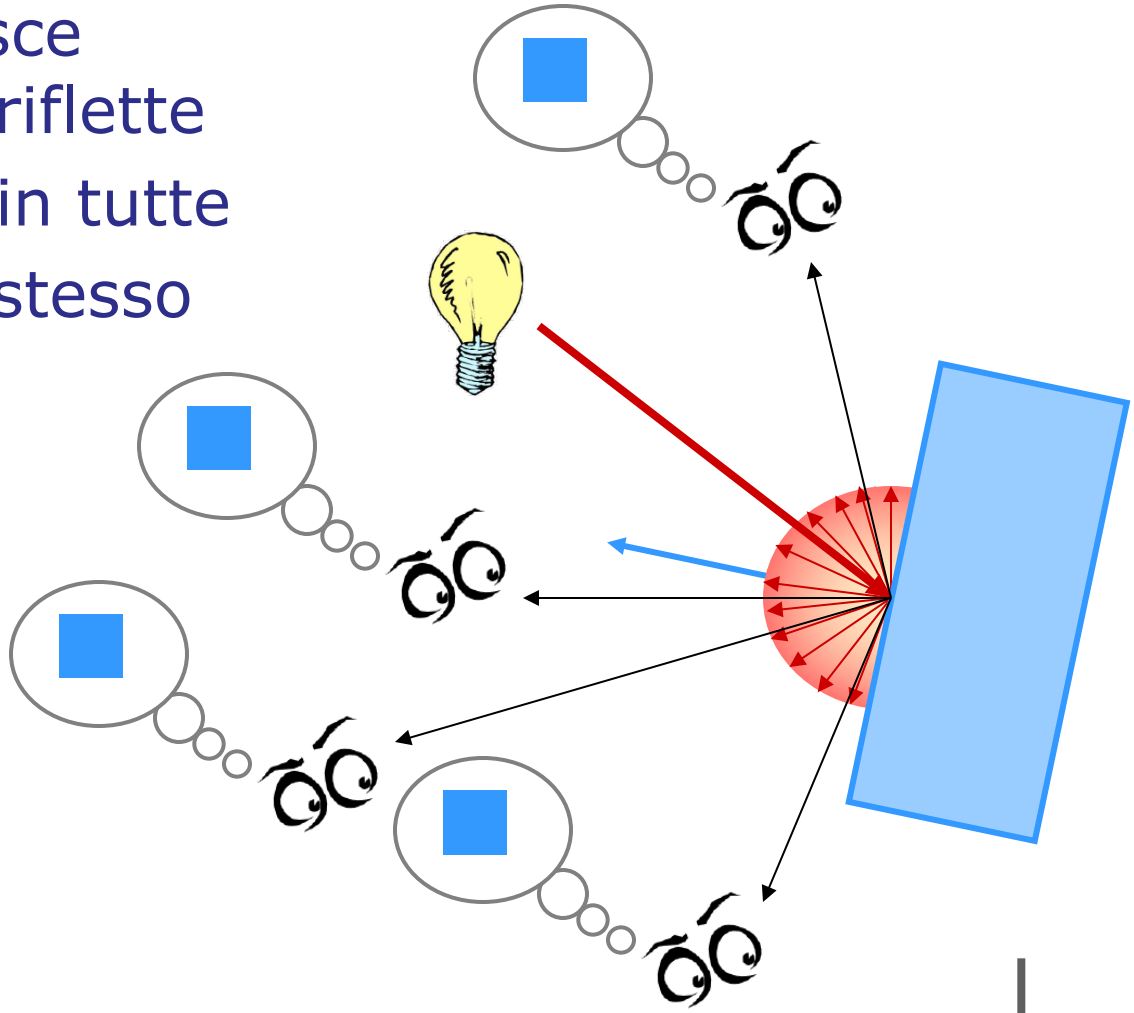




Componente *riflessione diffusa*

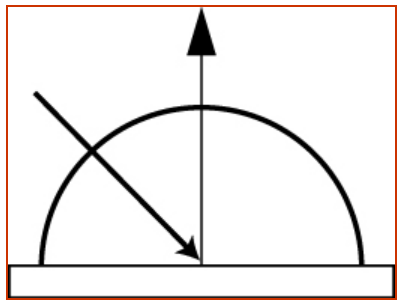
- La luce che colpisce una superficie si riflette (nel semispazio) in tutte le direzioni nello stesso modo

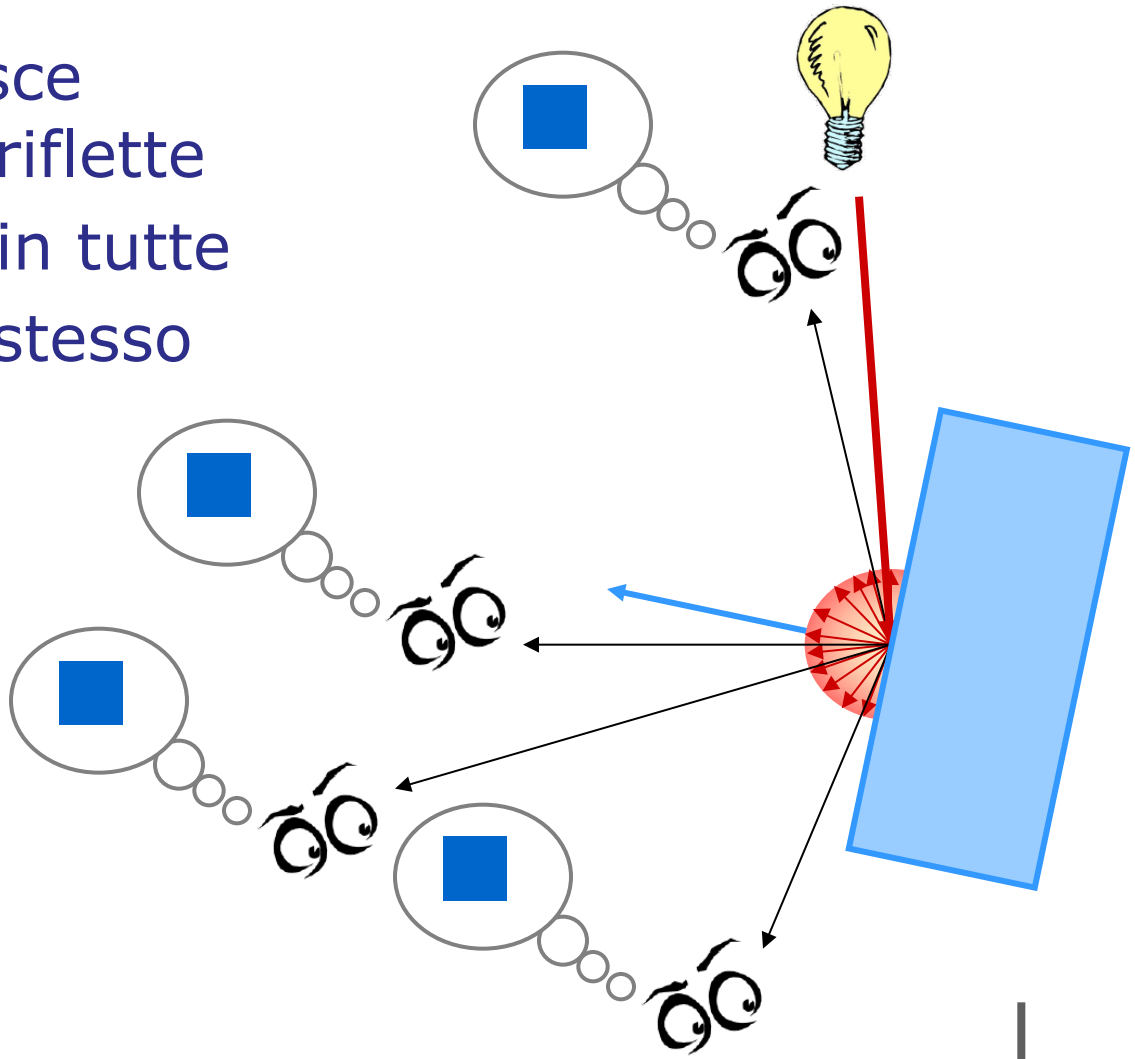




Componente *riflessione diffusa*

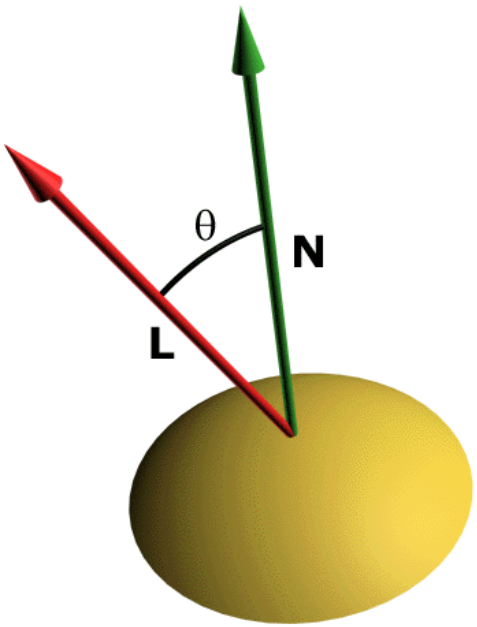
- La luce che colpisce una superficie si riflette (nel semispazio) in tutte le direzioni nello stesso modo





Componente *riflessione diffusa*

- Dipende da:
  - **N** normale alla superficie (orientamento della superficie)
  - **L** direzione della luce o del raggio incidente
  - θ angolo compreso



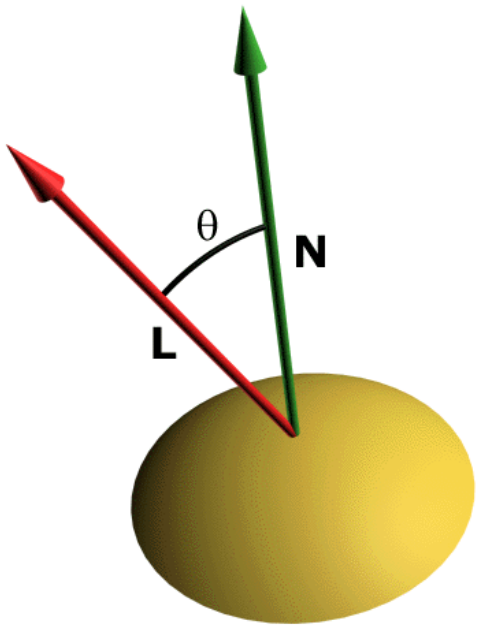
$$I_{dr} = k_d I_l \cos(\theta)$$

$K_R, K_G, K_B \in [0,1]$
(definisce il colore dell'oggetto)

R, G, B
(luce bianca: 1,1,1)

Componente *riflessione diffusa*

- Dipende da:
  - **N** normale alla superficie (orientamento della superficie)
  - **L** direzione della luce o del raggio incidente
  - θ angolo compreso

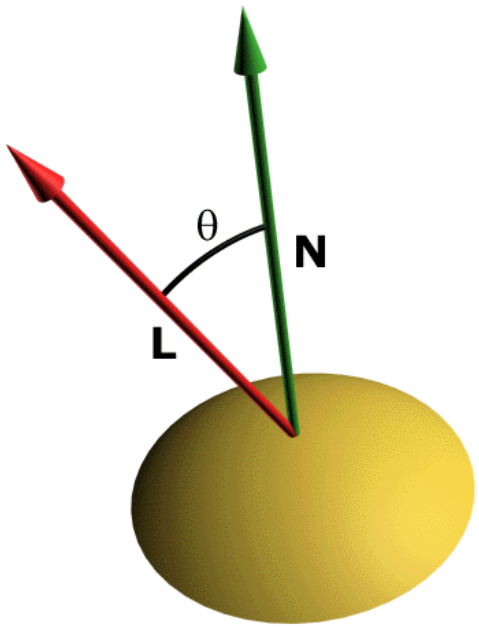


$$I_{dr} = k_d I_l \cos(\theta)$$

se l' angolo è compreso fra 0º e 90º,
altrimenti niente (oggetto in ombra di se stesso).

Componente *riflessione diffusa*

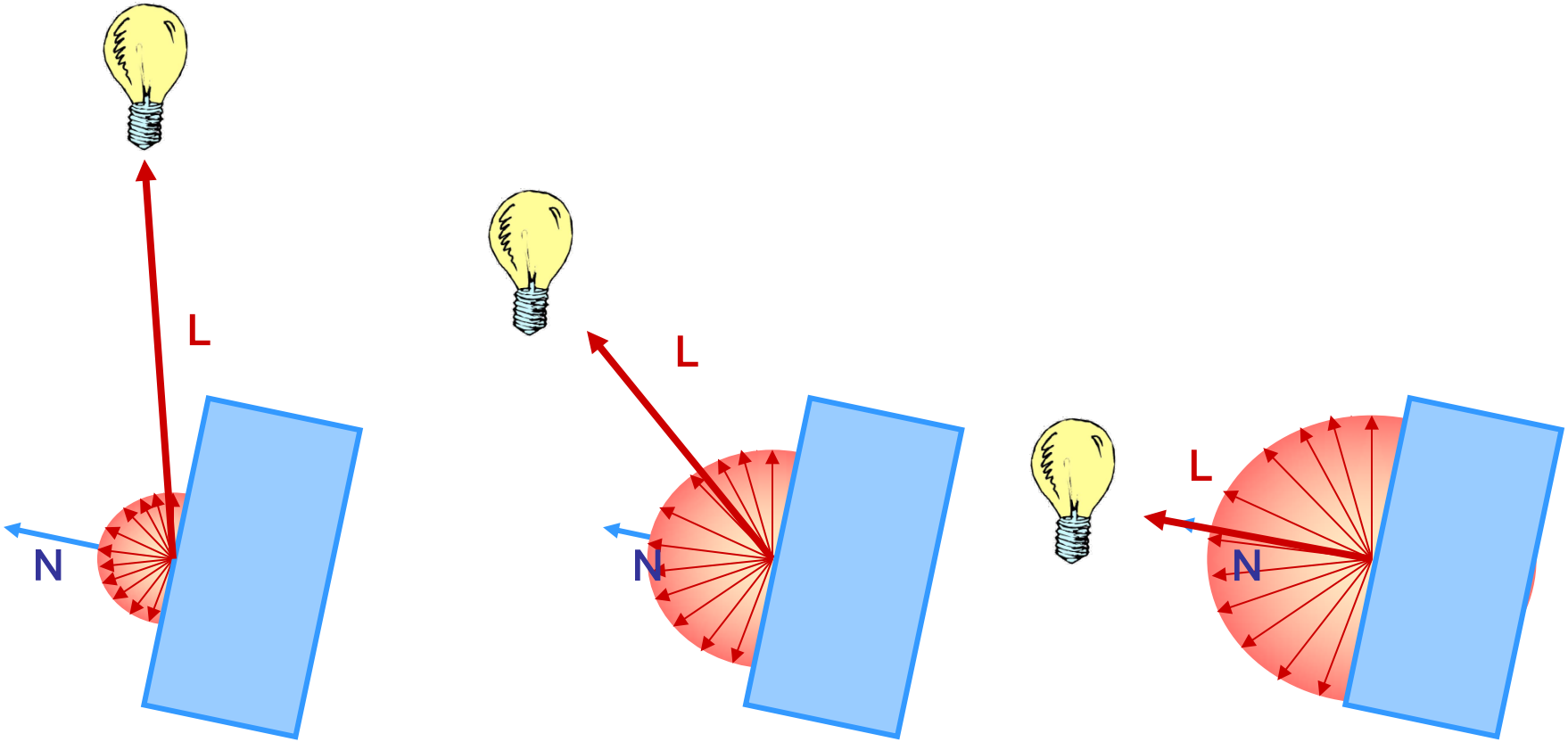
- Dipende da:
  - **N** normale alla superficie (orientamento della superficie)
  - **L** direzione della luce o del raggio incidente
  - θ angolo compreso



$$I_{dr} = k_d I_l \cos(\theta)$$

$$= k_d I_l (L \cdot N)$$

Componente *riflessione diffusa*

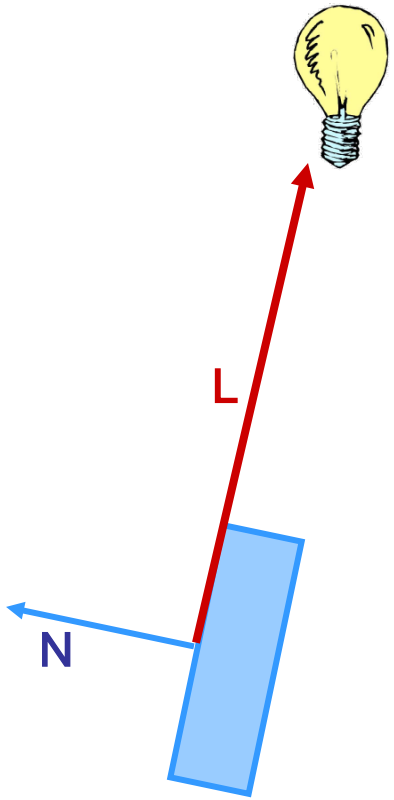


componente
diffusa
piccola
θ=70º

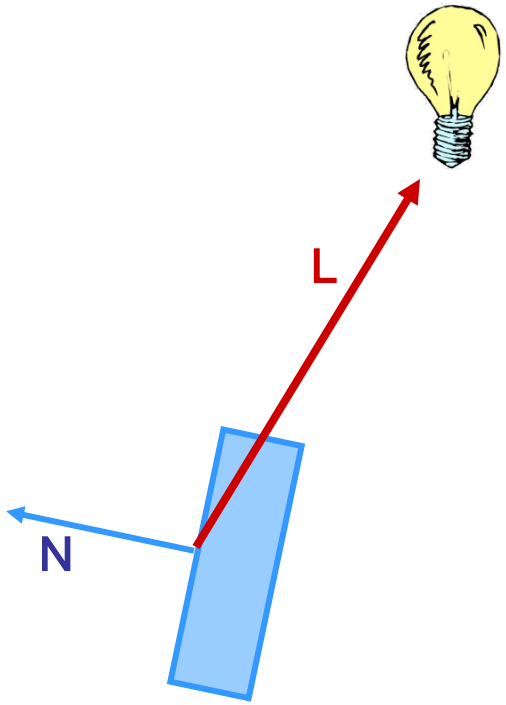
componente
diffusa
grande
θ=35º

componente
diffusa
massima
θ=0º

Componente *riflessione diffusa*



componente
diffusa
ZERO
θ=90º



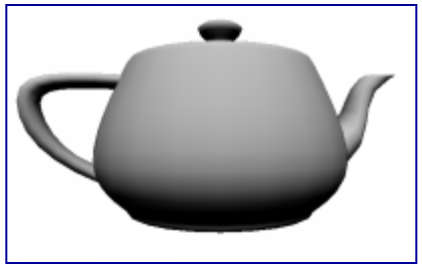
componente
diffusa
ZERO
θ>90º

la superficie
non è esposta
alla luce

Componente *riflessione speculare*

- E' tipica in superfici come per es.:
  - ceramica
  - metallo
  - …
- Per materiali lucidi
  - con riflessi brillanti (highlights)

Componente *ambiente*

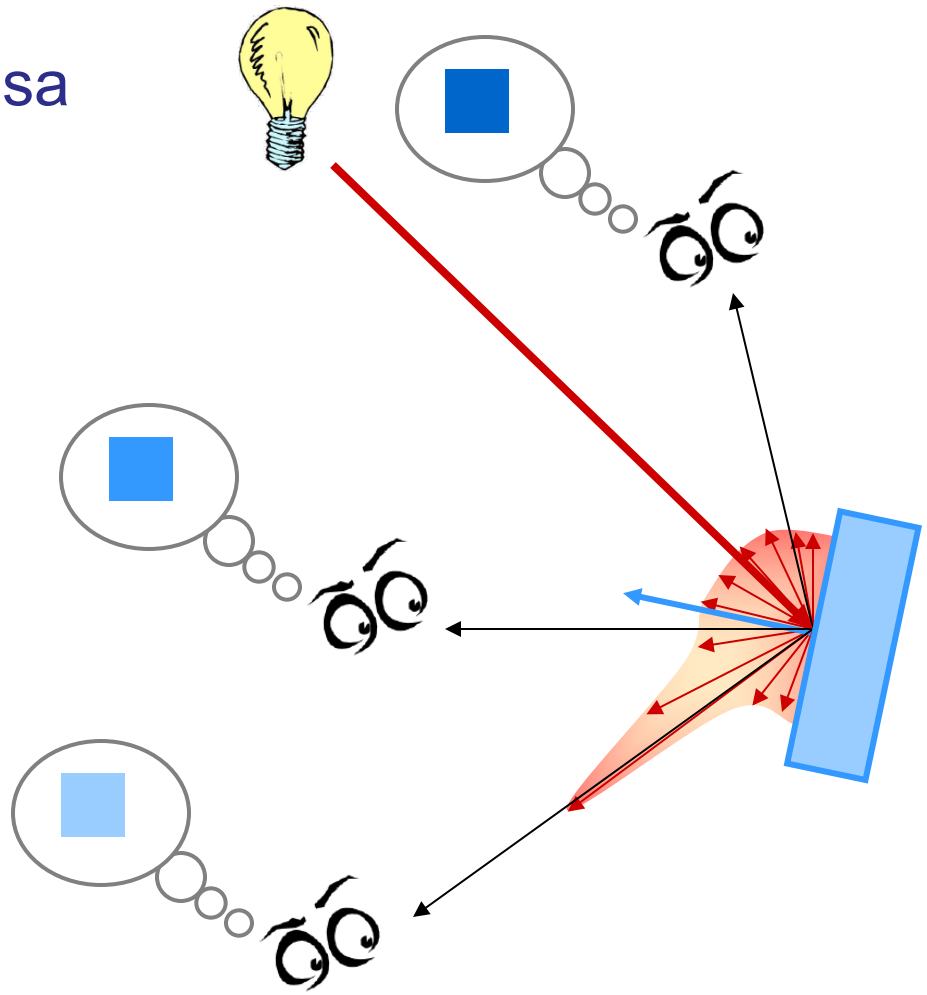


Componente *riflessione diffusa*

Componente *riflessione speculare*

Componente *riflessione speculare*

- Principio base:
  la luce non viene riflessa
  da materiali lucidi
  in maniera eguale
  in tutte le direzioni



Componente *riflessione speculare*

L: raggio incidente

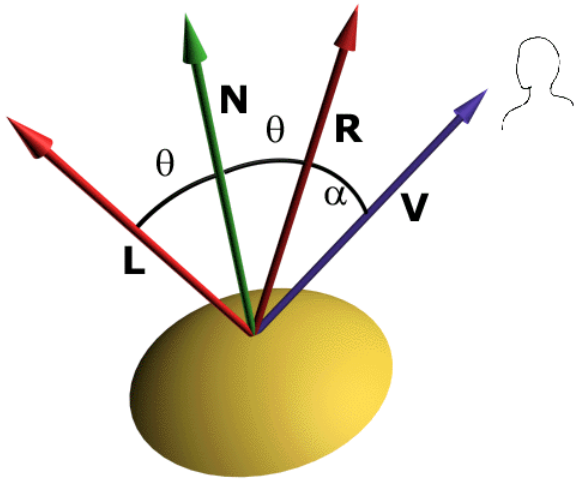
N: normale

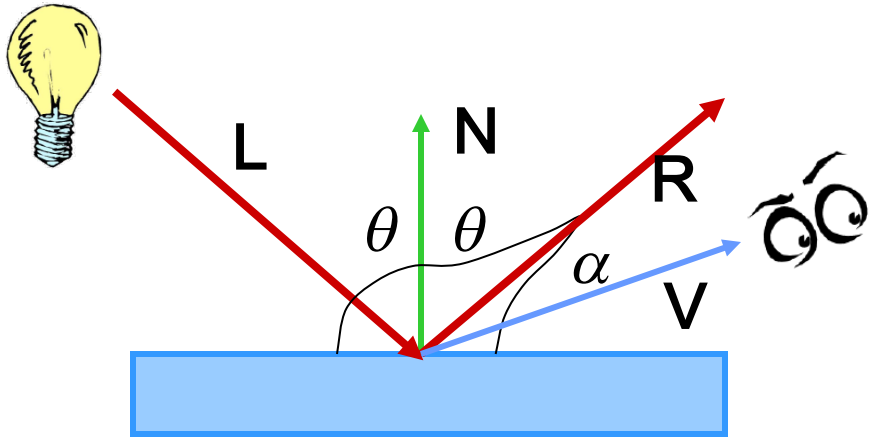
R: raggio riflesso

V: direzione di vista per vertice

θ: angolo fra L ed N

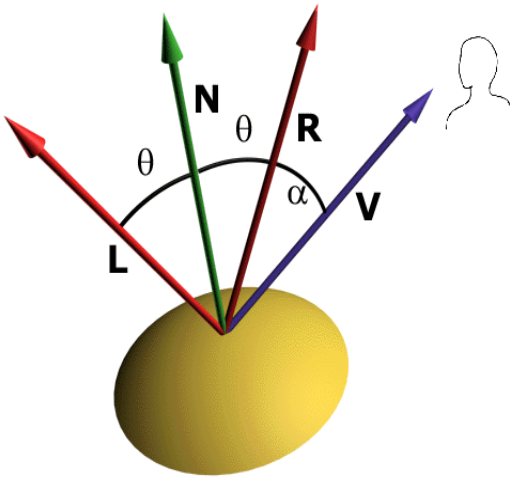
α: angolo fra R e V





Componente *riflessione speculare*

- Modello di Illuminazione di Phong
  (Bui-Tuong Phong, 1975)



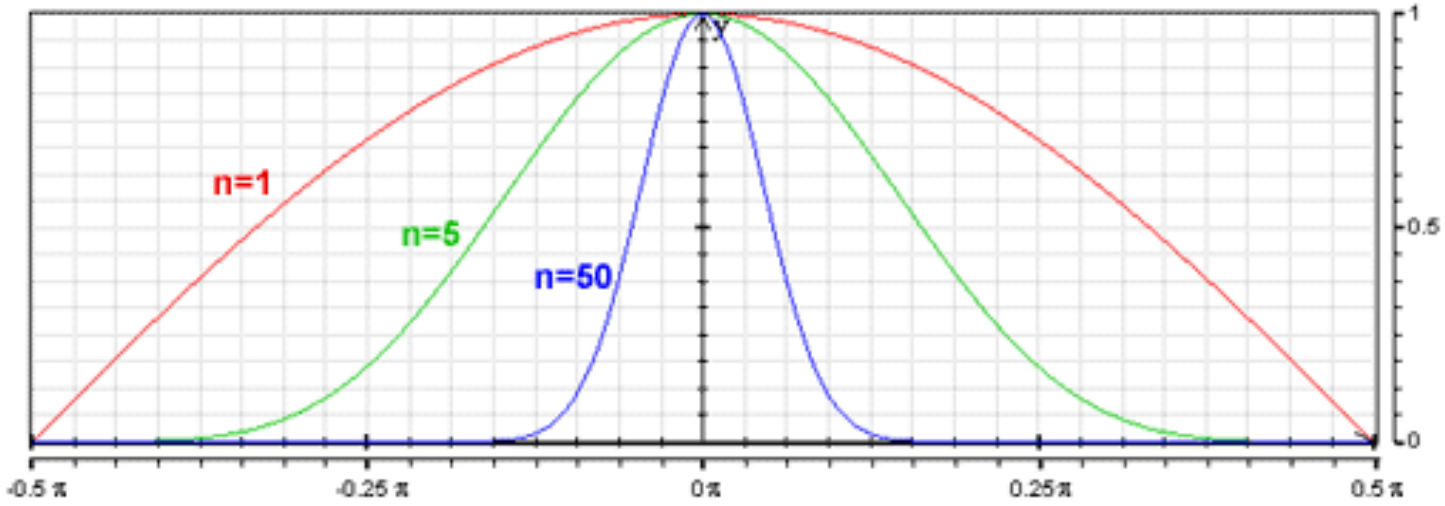
$$I_r = k_s \cdot I_l \cdot (\cos\alpha)^n$$

$$= k_s \cdot I_l \cdot (R \cdot V)^n$$

Vettore e Scalare caratteristici
del "materiale" dell'oggetto

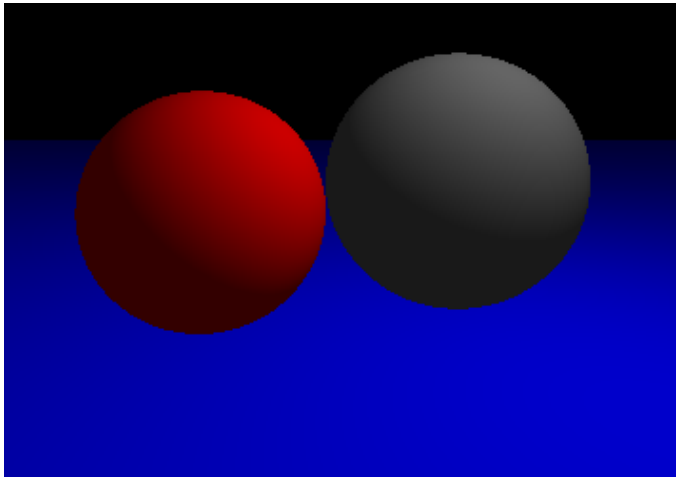
Componente *riflessione speculare*

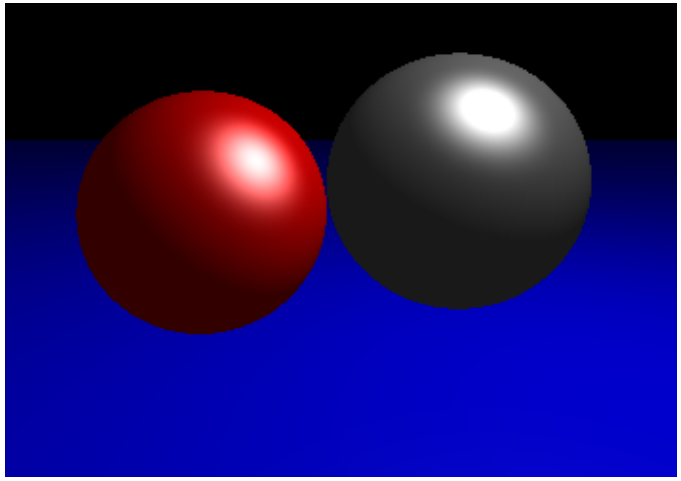
- Elevando il coseno ad una potenza,
  si ottengono riflessi più piccoli e brillanti



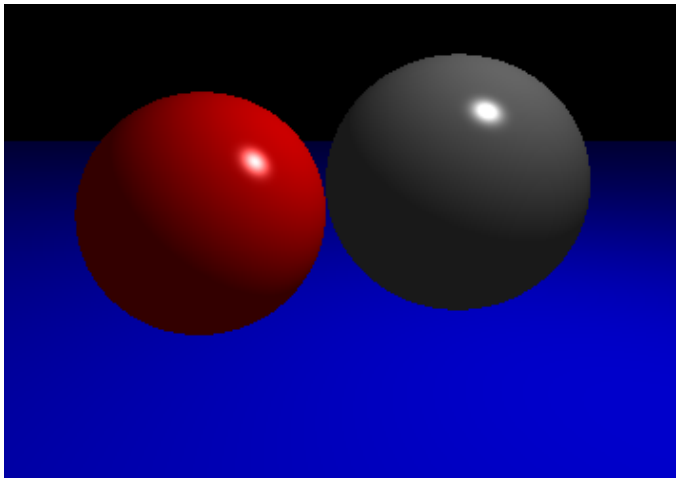
$$y = (\cos\alpha)^n$$

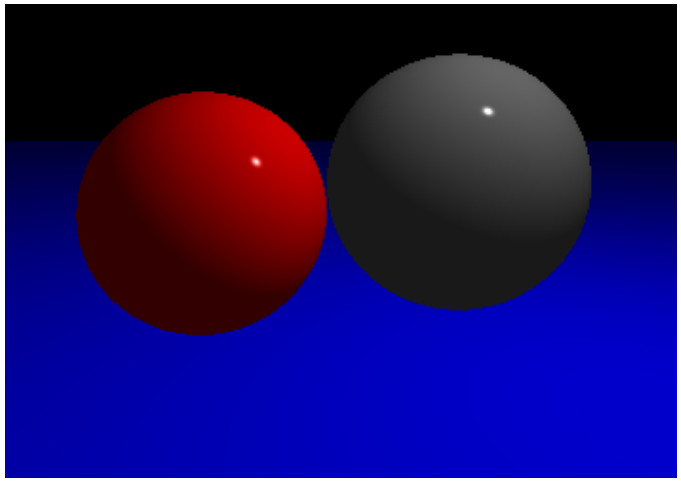
Componente *riflessione speculare*





$n = 5$





$n = 10$

$n = 500$

Equazione di lighting totale

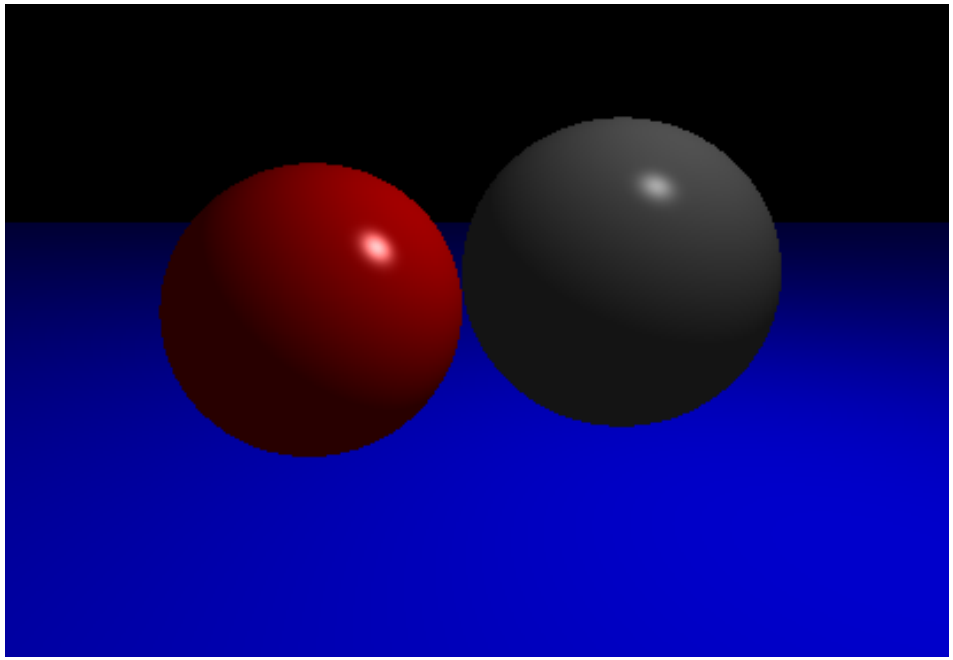
$$I = k_a I_a + I_l \left(k_d (L \cdot N) + k_s (R \cdot V)^n \right)$$

propretà della luce

propretà del materiale

Equazione di lighting totale

$$I = k_a I_a + I_l \left(k_d (L \cdot N) + k_s (R \cdot V)^n \right)$$

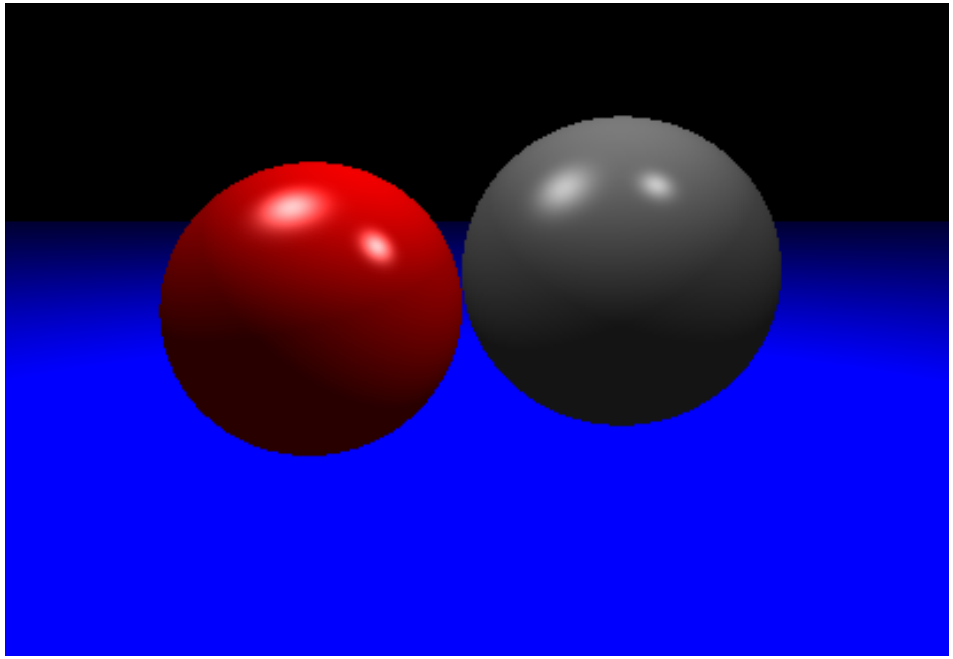


Equazione di lighting totale

$$I = k_a I_a + I_l \left(k_d (L \cdot N) + k_s (R \cdot V)^n \right)$$

$$I = k_a I_a + \sum_i I_{l_i} \left(k_d (L_i \cdot N) + k_s (R_i \cdot V)^n \right)$$

Più sorgenti luminose



Materiali...

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR
Emerald	0.0215 0.1745 0.0215 0.55	0.07568 0.61424 0.07568 0.55	0.633 0.727811 0.633 0.55
Jade	0.135 0.2225 0.1575 0.95	0.54 0.89 0.63 0.95	0.316228 0.316228 0.316228 0.95
Obsidian	0.05375 0.05 0.06625 0.82	0.18275 0.17 0.22525 0.82	0.332741 0.328634 0.346435 0.82
Pearl	0.25 0.20725 0.20725 0.922	1.0 0.829 0.829 0.922	0.296648 0.296648 0.296648 0.922
Ruby	0.1745 0.01175 0.01175 0.55	0.61424 0.04136 0.04136 0.55	0.727811 0.626959 0.626959 0.55
Turquoise	0.1 0.18725 0.1745 0.8	0.396 0.74151 0.69102 0.8	0.297254 0.30829 0.306678 0.8
Black Plastic	0.0 0.0 0.0 1.0	0.01 0.01 0.01 1.0	0.50 0.50 0.50 1.0
Black Rubber	0.02 0.02 0.02 1.0	0.01 0.01 0.01 1.0	0.4 0.4 0.4 1.0
Brass	0.329412 0.223529 0.027451 1.0	0.780392 0.568627 0.113725 1.0	0.992157 0.941176 0.807843 1.0
Bronze	0.2125 0.1275 0.054 1.0	0.714 0.4284 0.18144 1.0	0.393548 0.271906 0.166721 1.0
Polished Bronze	0.25 0.148 0.06475 1.0	0.4 0.2368 0.1036 1.0	0.774597 0.458561 0.200621 1.0
Chrome	0.25 0.25 0.25 1.0	0.4 0.4 0.4 1.0	0.774597 0.774597 0.774597 1.0

k_a k_d k_s

Materiali nei file in formato .obj

https://en.wikipedia.org/wiki/Wavefront_.obj_file

```
mtllib pippo.mtl
…
usemtl red_mat
f 2096 2097 2084
…
usemtl blue_mat
f 2076 2075 2104
…
```

file.obj

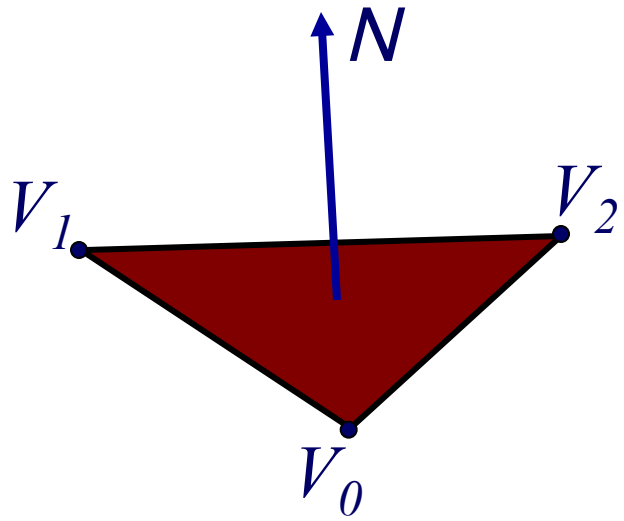
```
newmtl red_mat
Ka 0.4000 0.4000 0.4000
Kd 0.3000 0.0000 0.0000
Ks 0.3000 0.3000 0.3000
illum 2
Ns 60.0000

newmtl blue_mat
Ka 0.4000 0.4000 0.4000
Kd 0.0000 0.0000 0.3000
Ks 0.3000 0.3000 0.3000
illum 2
Ns 60.0000
```

pippo.mtl

Potenza n nel modello di Phong

Normale di un triangolo

- Cioè il suo orientamento nello spazio

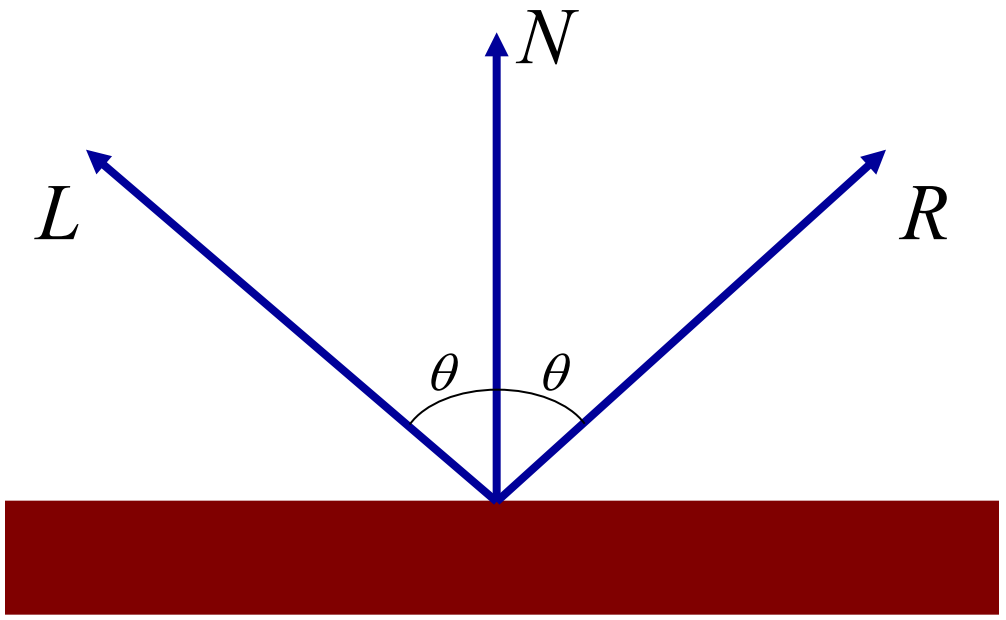


$$N = (V_2 - V_0) \times (V_1 - V_0)$$

$$\hat{N} = \frac{N}{\| N \|}$$

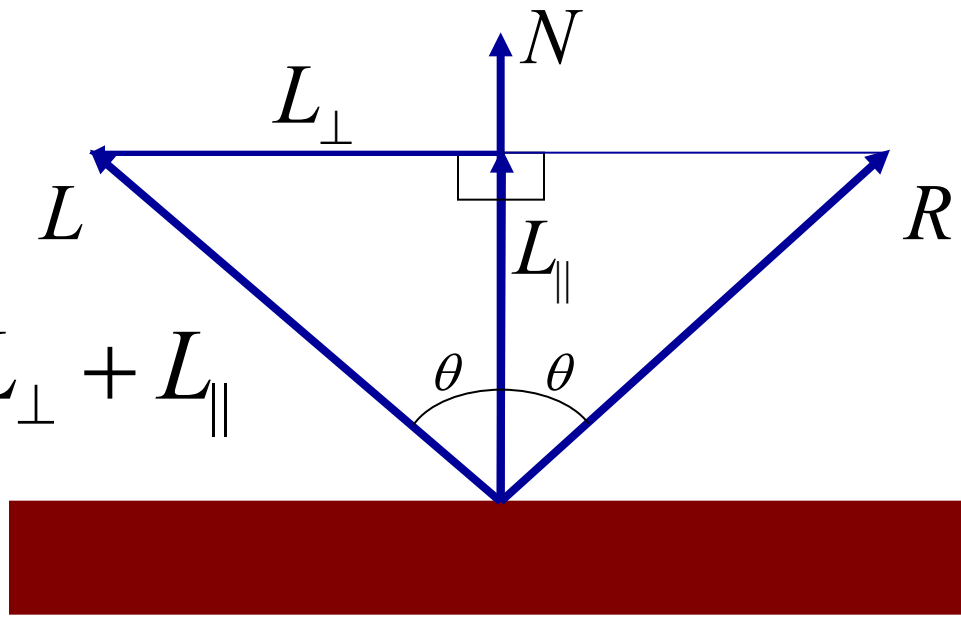
Ancora su *riflessione speculare*

- Come si calcola il vettore riflesso R ?



Ancora su *riflessione speculare*

- Come si calcola il vettore riflesso R ?



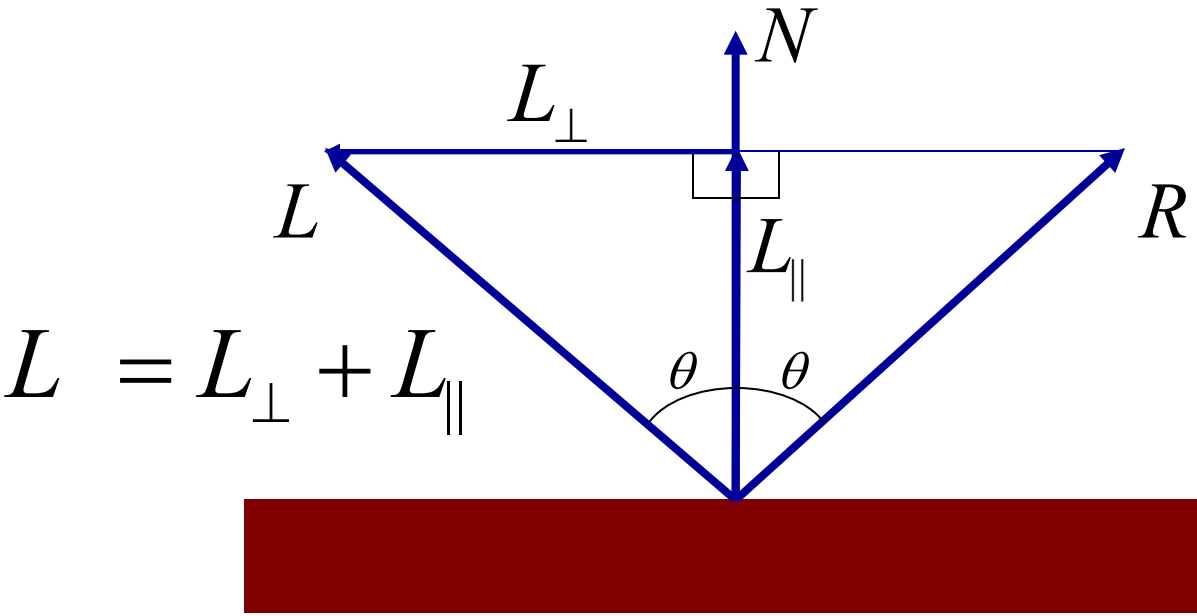
$$L = L_\perp + L_\parallel$$

Ancora su *riflessione speculare*

- Come si calcola il vettore riflesso R ?

$$L_{\parallel} = N\cos(\theta) = N(L \cdot N)$$

$$L_{\perp} = L - L_{\parallel}$$

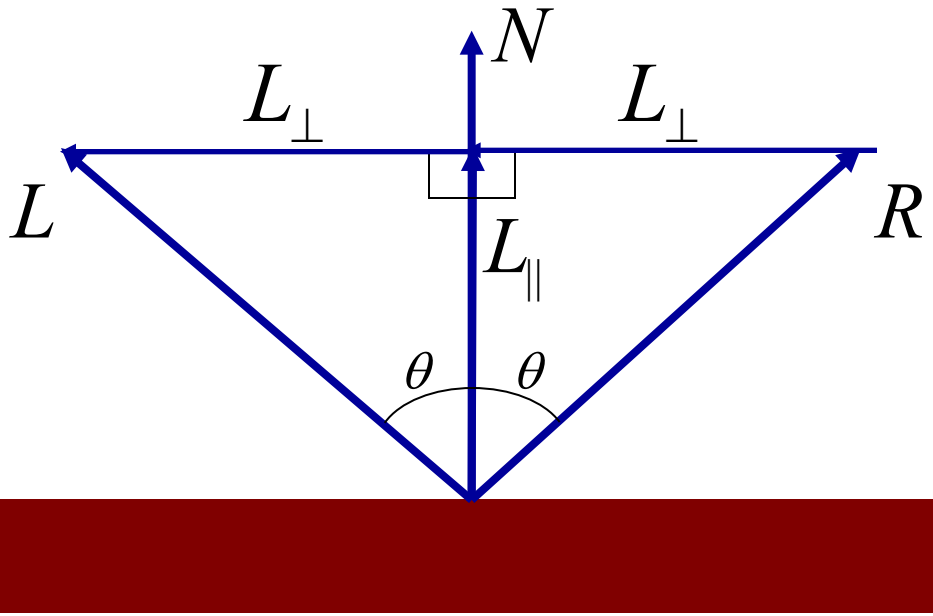


$$L = L_{\perp} + L_{\parallel}$$

Ancora su *riflessione speculare*

- Come si calcola il vettore riflesso R ?

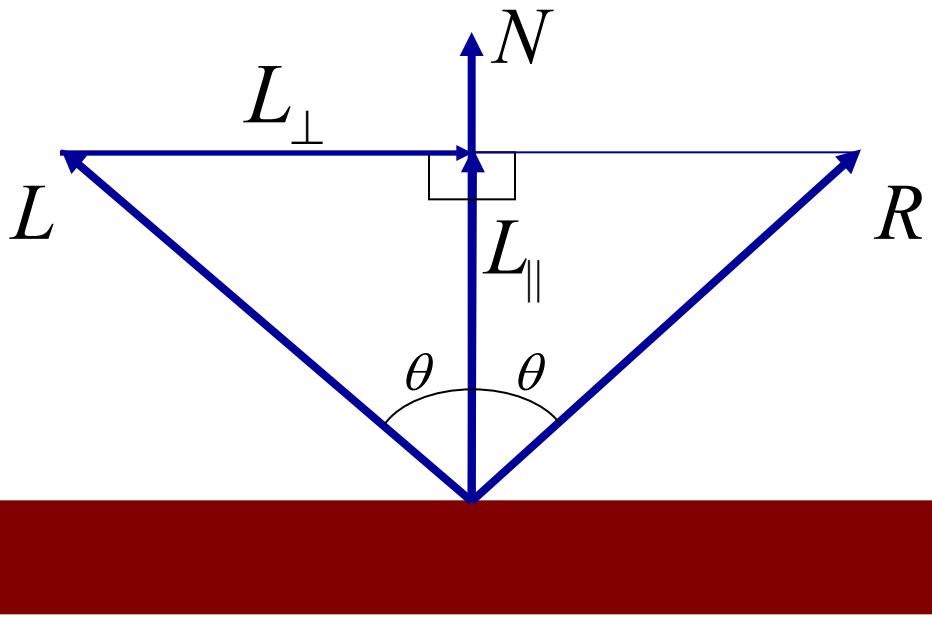
$$R = L_{\parallel} - L_{\perp}$$



Ancora su *riflessione speculare*

- Come si calcola il vettore riflesso R ?

$$R = L_{\parallel} - L_{\perp}$$
$$= 2(L \cdot N)N - L$$



Ancora su *riflessione speculare*

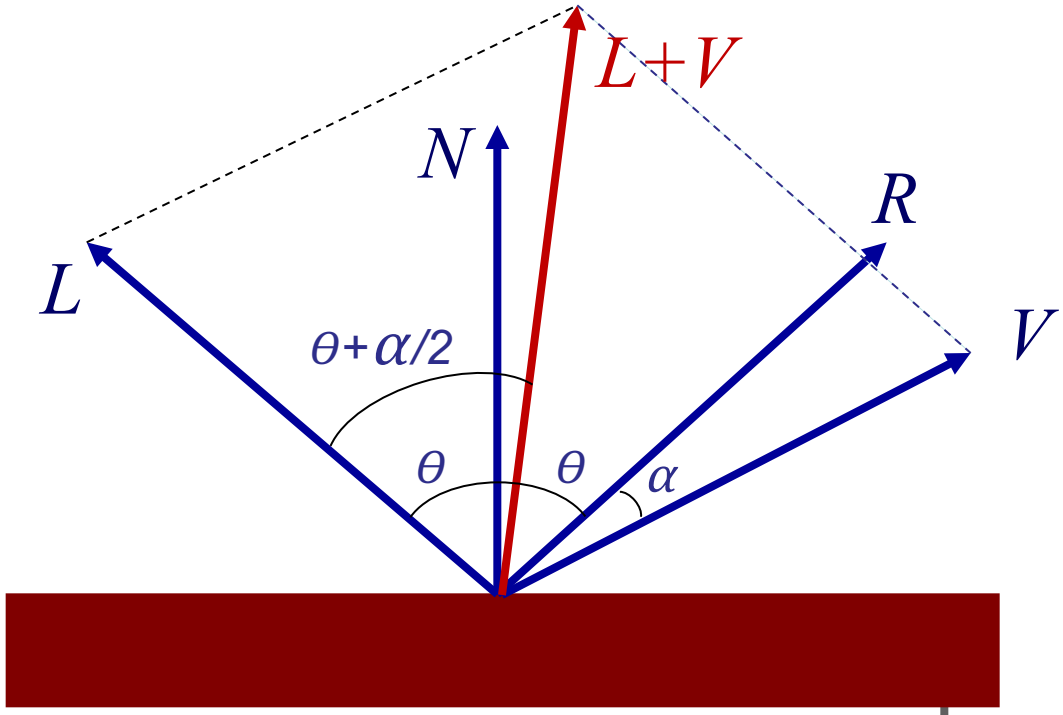
- Una volta determinato R si deve calcolare $R \cdot V$ ed essendo vettori normalizzati il loro prodotto scalare equivale al coseno dell'angolo α compreso.
- $L+V$ è un vettore che forma con N un angolo α/2.

- Allora possiamo usare

  $\cos(\alpha/2)^n$

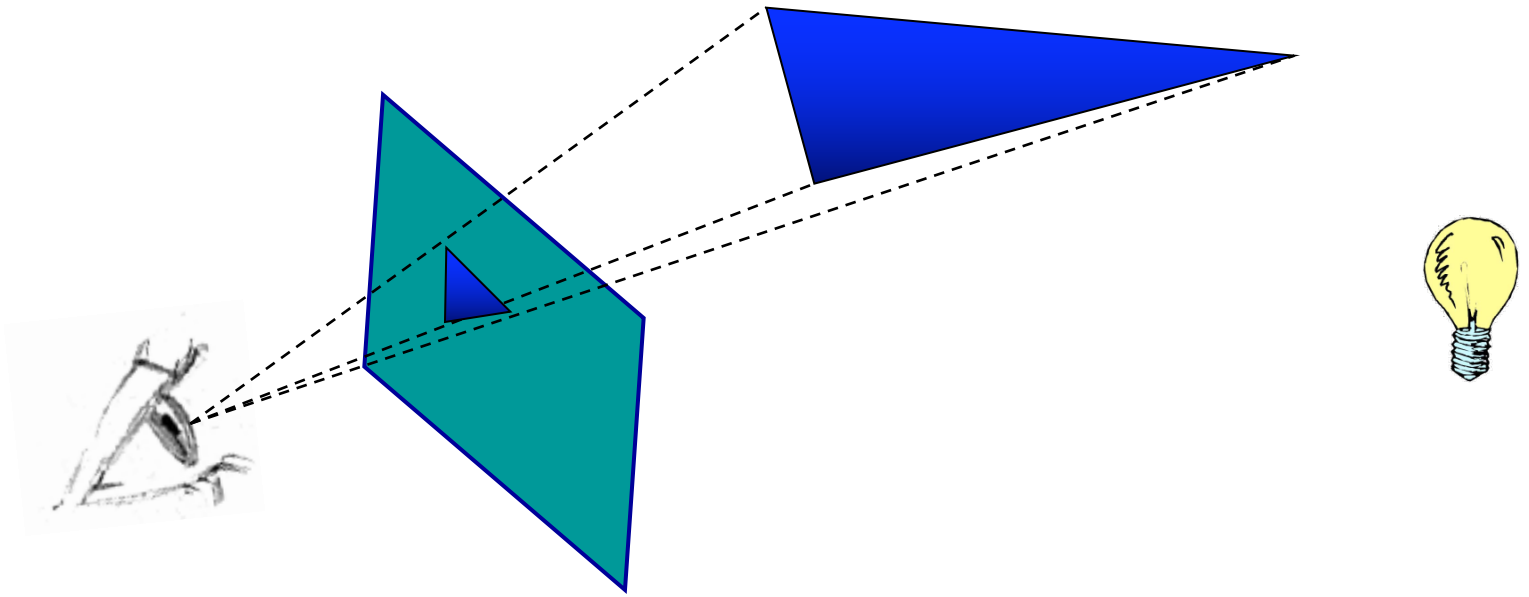
come termine nel modello di Phong al posto di

$(R \cdot V)^n = \cos(\alpha)^n$



Problema

- Come si colora un triangolo illuminato?
  mediante rasterizzazione con una specifica Tecnica o Algoritmo di Shading



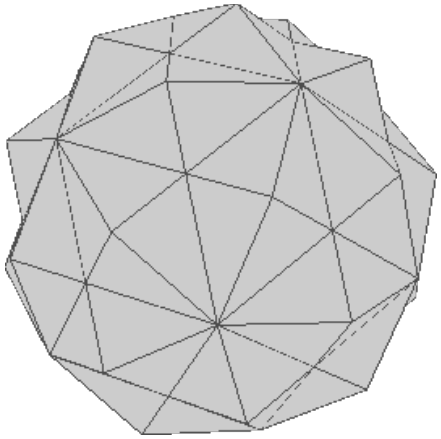
Algoritmi di Shading

•Vediamo le seguenti semplici Tecniche di Shading:

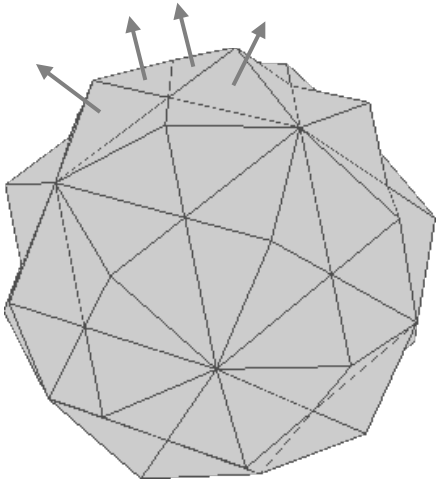
- Flat Shading
- Gouraud Shading
- Phong Shading

Flat Shading:

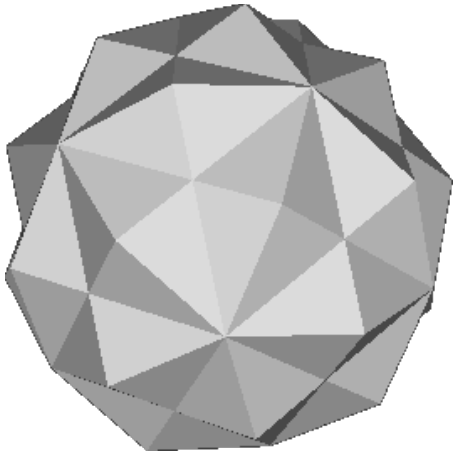
Illuminazione faccia per faccia



1. geometria di partenza



2. per ogni faccia, si calcola la normale



3. si applica il modello di illuminazione al centro di ogni faccia e si calcola un colore per faccia

Flat Shading: problema

Le superfici curve vengono approssimate con triangoli (tassellazione), poi si applica l'algoritmo di Flat Shading ad ogni triangolo

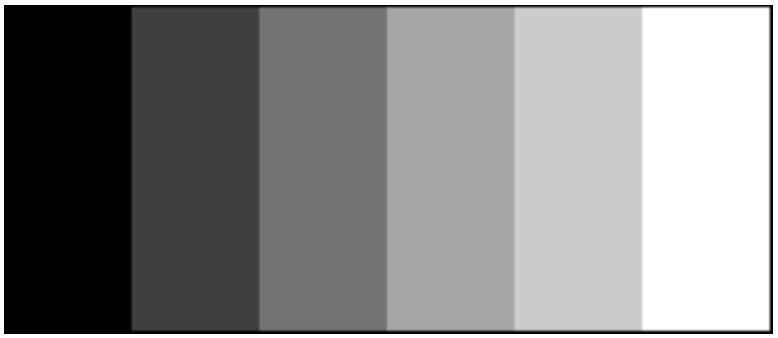
Risultato:

spigoli evidenti: un risultato mediocre!

Flat Shading: problema

A peggiorare le cose c'è l'effetto ottico noto come:

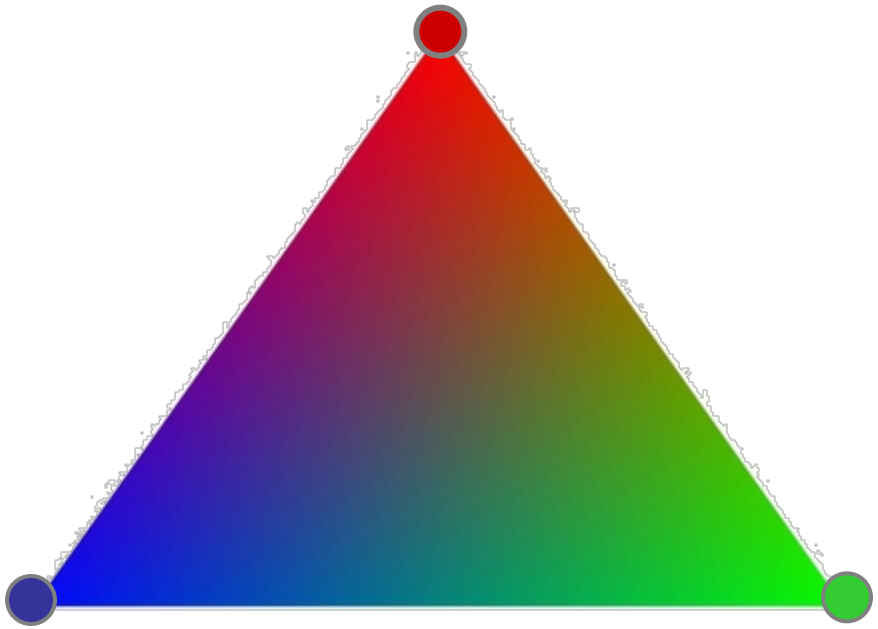
Mach-band



Il contrasto fra zone di colore uniforme difficilmente sfugge al nostro occhio (neanche se le zone sono molte, e la differenza fra loro è relativamente piccola).
Il cervello umano aumenta il contrasto fra zone di colore uniforme.

Idea già nota

Utilizziamo l'interpolazione colore dei vertici



Gouraud Shading (Henri Gouraud, 1971)

Utilizzare l'interpolazione del colore dei vertici

1- Si applica il modello di illuminazione ai tre vertici di ogni triangolo; questo permette di determinare un colore per ciascun vertice

2- Si interpolano i tre colori per ogni pixel nel triangolo (componente colore per componente colore)

Per applicare il modello di illuminazione serve la normale!
Normale definita per vertice (non per faccia)!

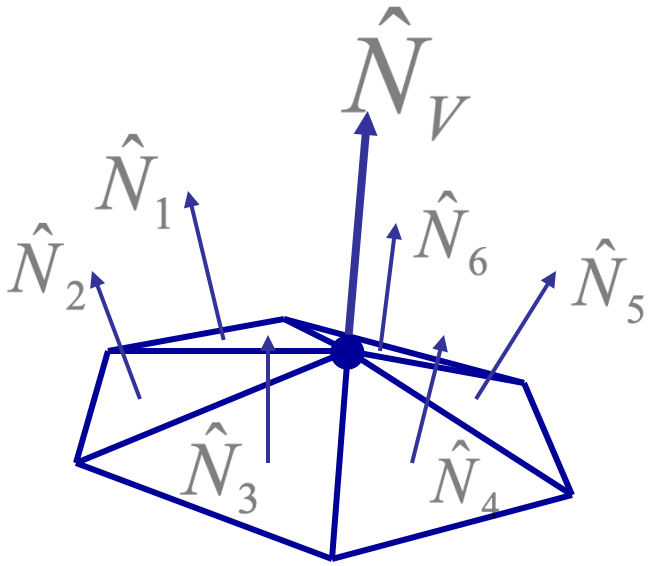
Normali per vertice

Normale per un vertice condiviso da *n* triangoli:

$$N = \hat{N}_1 + \hat{N}_2 + \ldots + \hat{N}_n$$

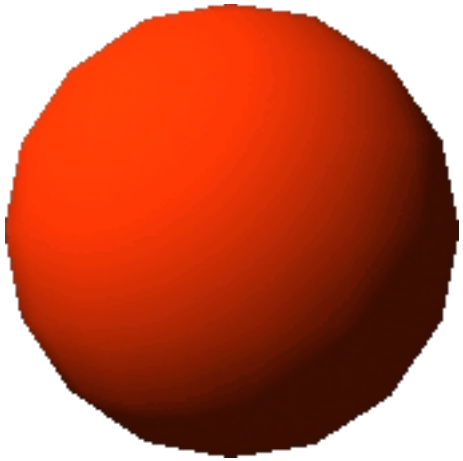
Quindi si normalizza

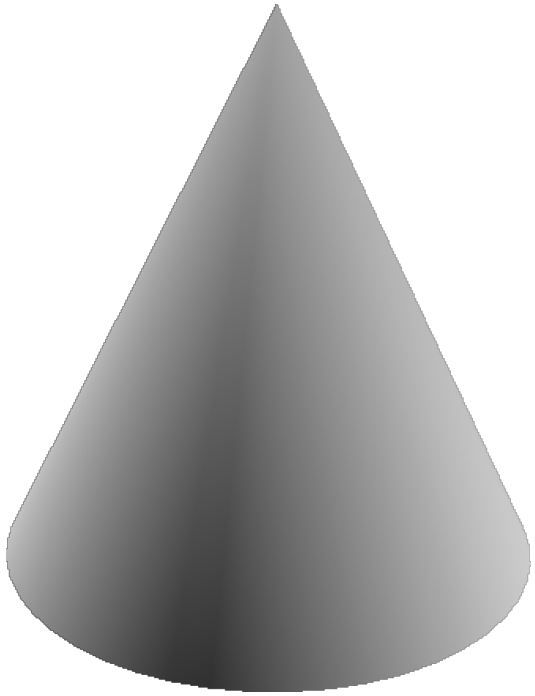
$$\hat{N}_V = \frac{N}{\| N \|}$$

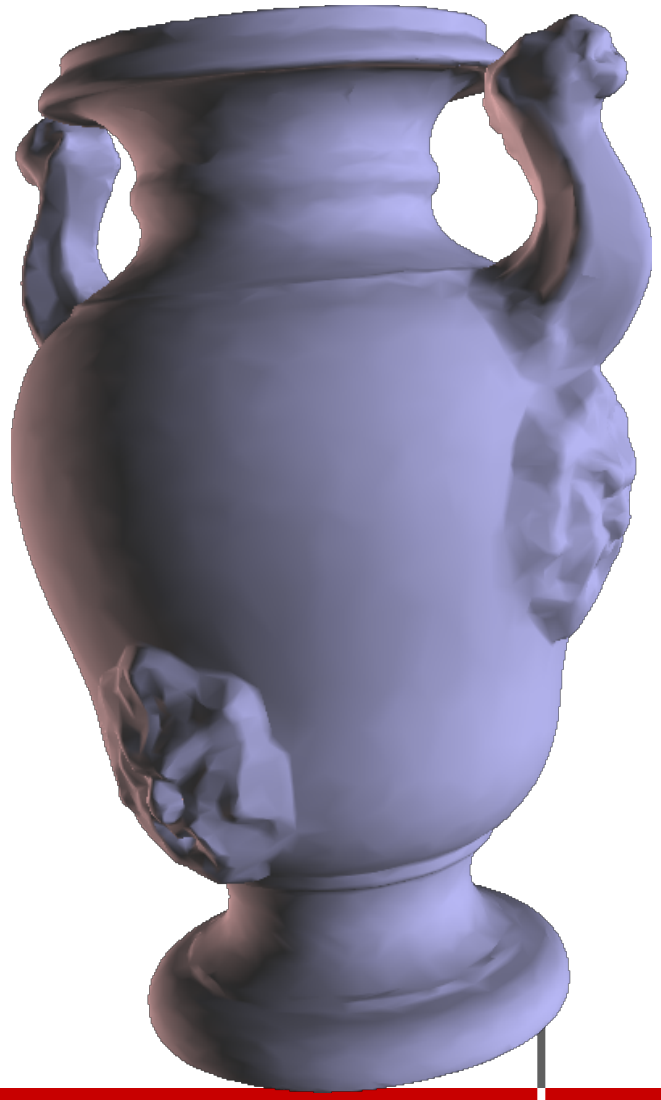


Gouraud Shading

Risultati:

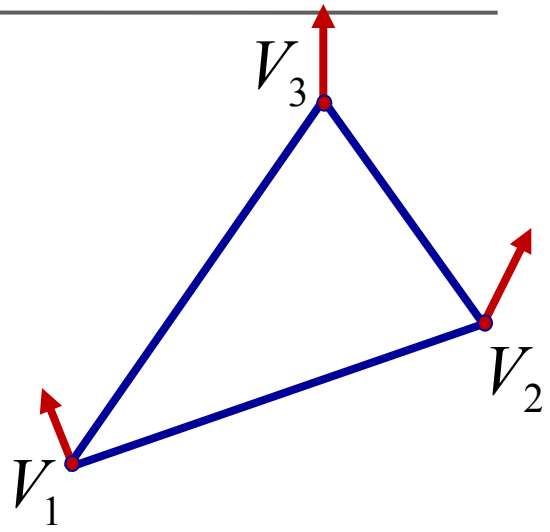






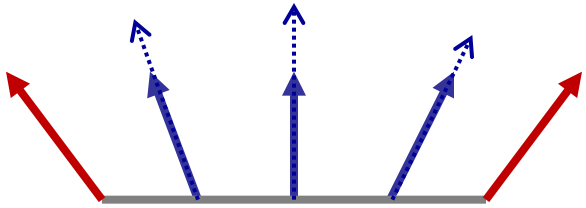
Si può fare meglio?

- Invece di interpolare il colore *dopo* l'applicazione del modello di illuminazione nei tre vertici, si interpola la normale nei tre vertici *prima* di applicare l'illuminazione!



- Attenzione:
  interpolando due vettori normali, non si ottiene un vettore normalizzato:

  si deve rinormalizzare dopo l'interpolazione



Phong Shading (Bui-Tuong Phong, 1973)

1- Si interpolano le normali nei tre vertici per ottenere differenti normali nella faccia

2- Si rinormalizzano

3- Si applica il modello di illuminazione ad ogni punto

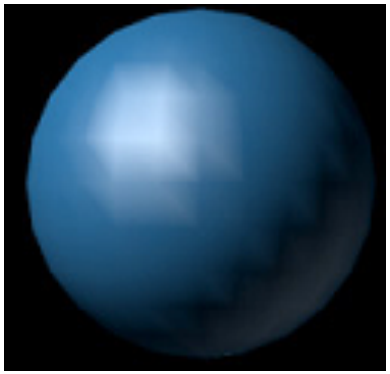
Attenzione, non confondere il Phong Shading (un algoritmo di shading) con il Phong Lighting Model (il modello di illuminazione)

Gouraud vs Phong Shading

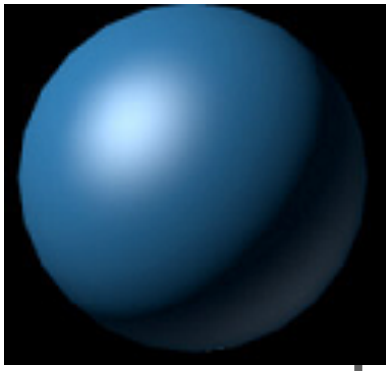
- Gouraud Shading - illuminazione per vertice;
  molto veloce:
  si applica l'illuminazione una volta per vertice!

- Phong Shading - illuminazione per "frammento";
  risultati migliori, ma costoso
  ottimo per riflessi luminosi (esponente speculare alto)

Flat Shading

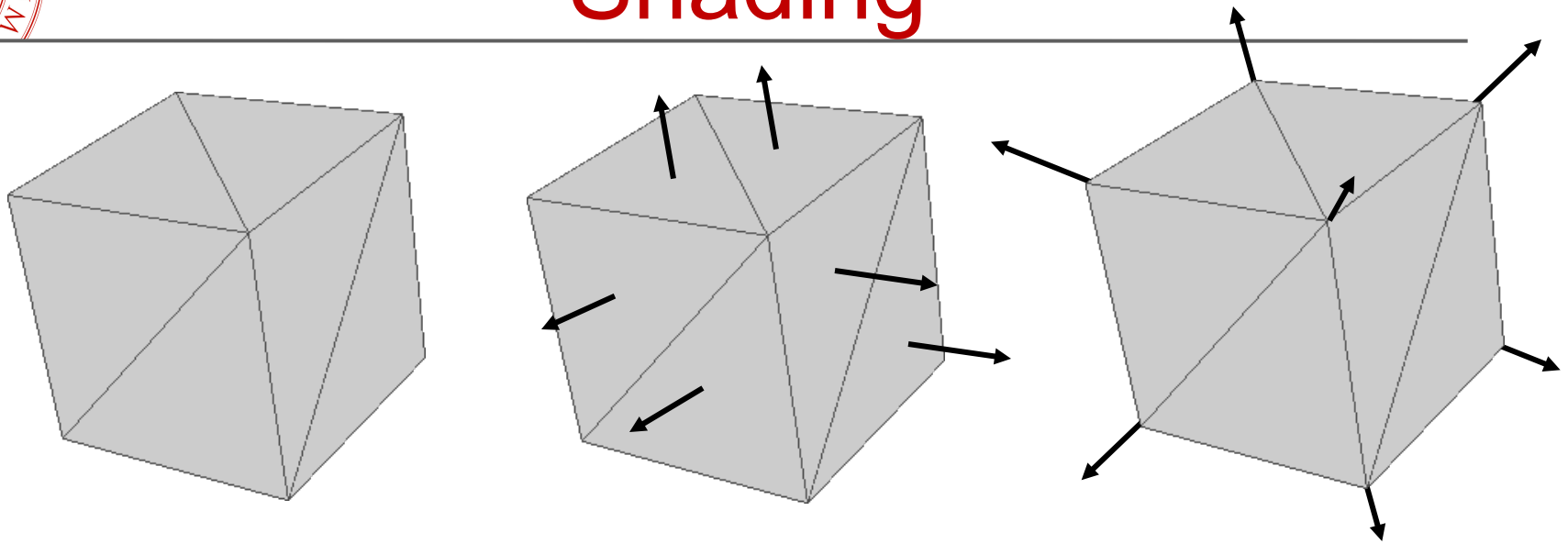


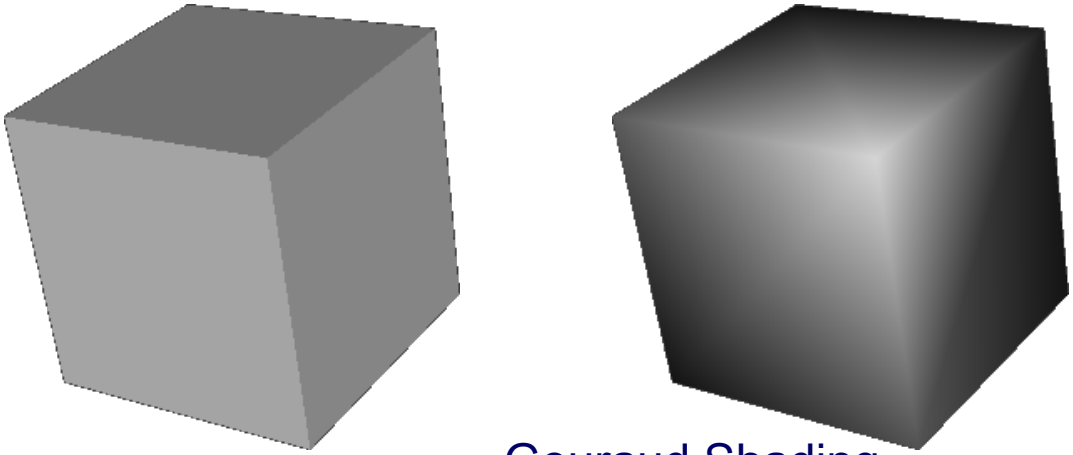
Gouraud Shading



Phong Shading

Note per Gouraud e Phong Shading



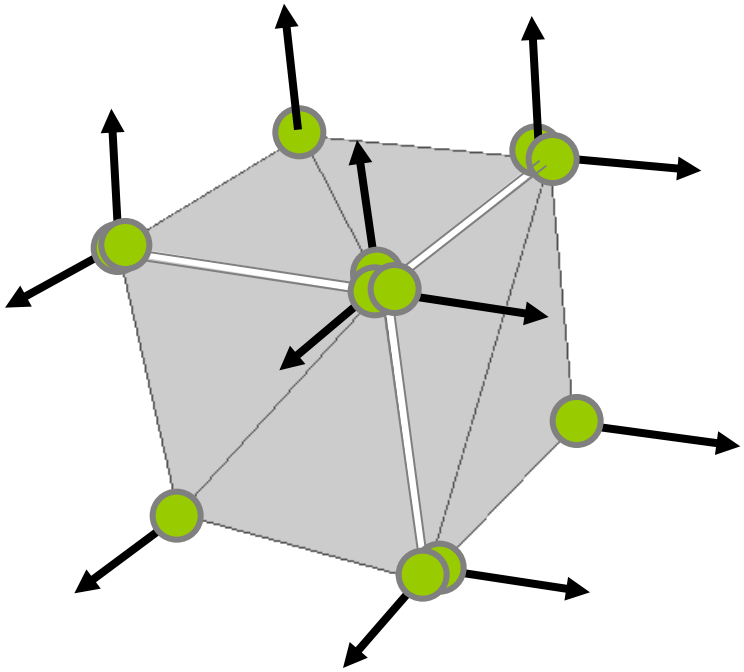


Flat Shading

Gouraud Shading
(Phong Shading in questso caso è simile)

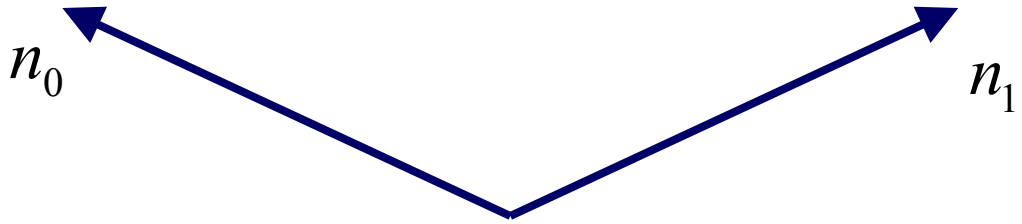
Note per Gouraud e Phong Shading

- Gouraud e Phong servono per superfici lisce
  - eliminano gli spigoli artefatti
  - ma eliminano anche gli spigoli vivi!
- in questo ultimo caso duplicare i vertici



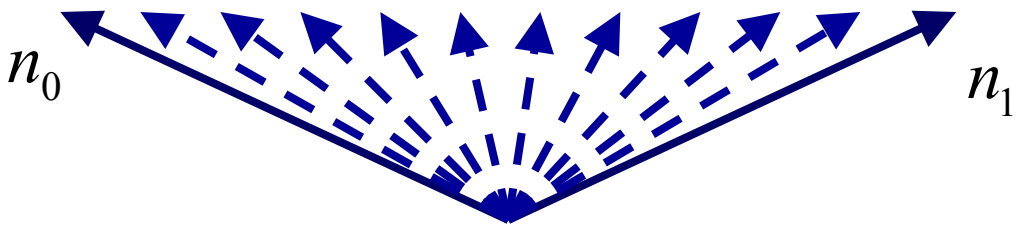
Nota sull' interpolazione delle Normali

- Si procede come per il colore (componente per componente)
- Si deve rinormalizzare
- Ma, non produce normali equispaziate!



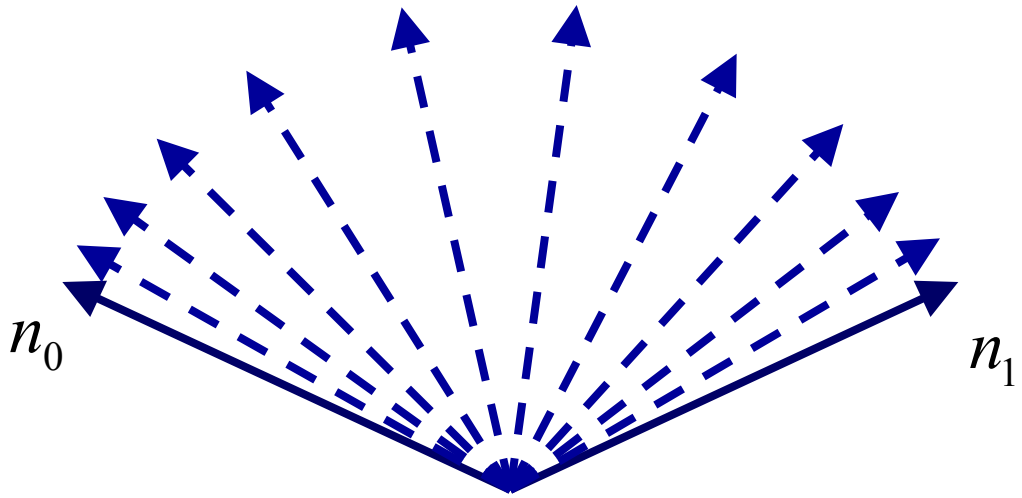
Interpolazione delle Normali

- Si procede come per il colore (componente per componente)
- Si deve rinormalizzare
- Ma, non produce normali equispaziate!

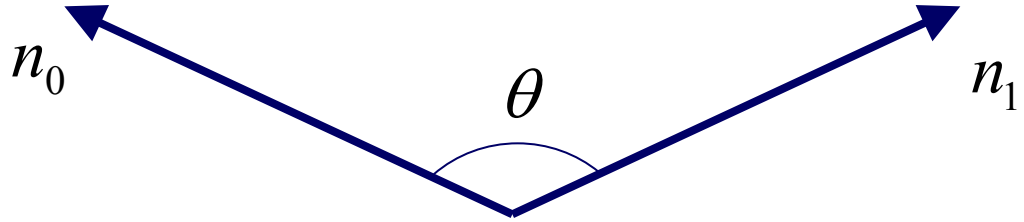


Interpolazione delle Normali

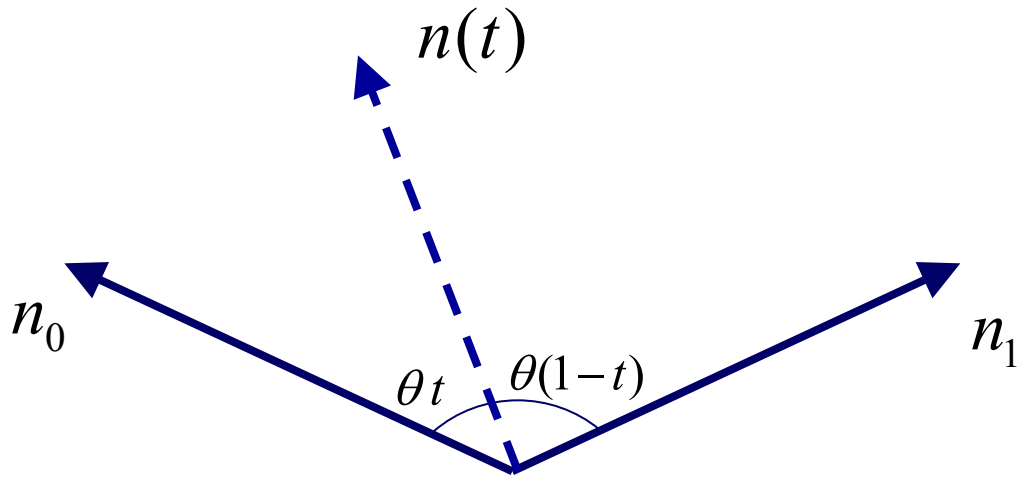
- Si procede come per il colore (componente per componente)
- Si deve rinormalizzare
- Ma, non produce normali equispaziate!



SLERP
(Spherical Linear Interpolation)

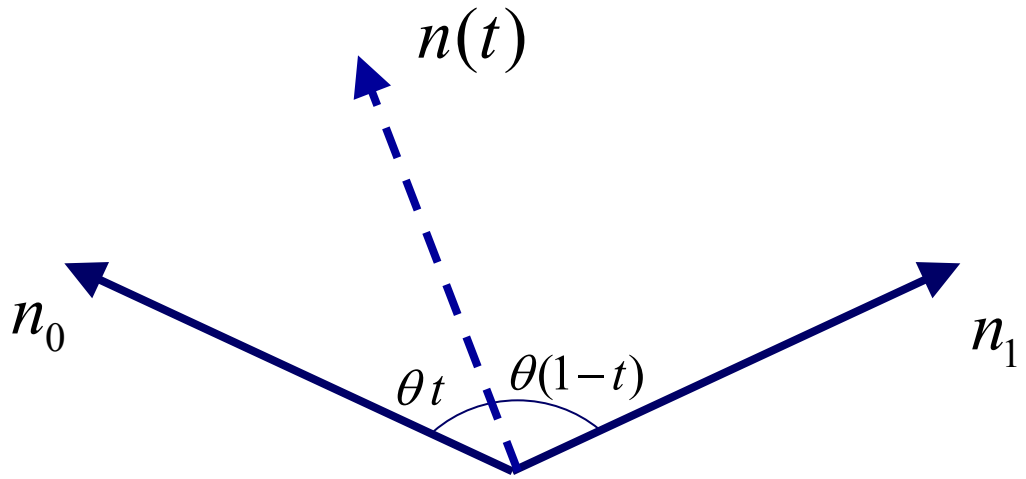


SLERP
(Spherical Linear Interpolation)



SLERP
(Spherical Linear Interpolation)

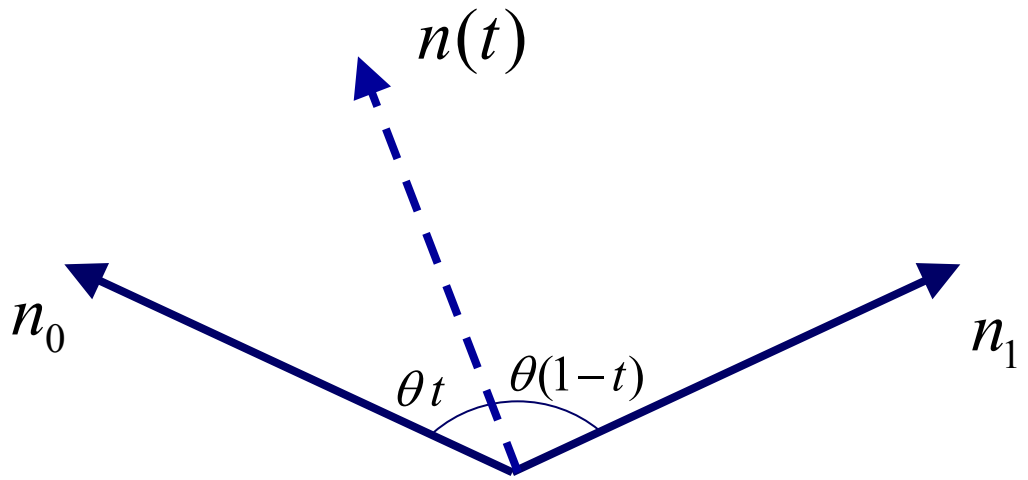
$$| n_0 | = | n_1 | = | n(t) |$$



SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$



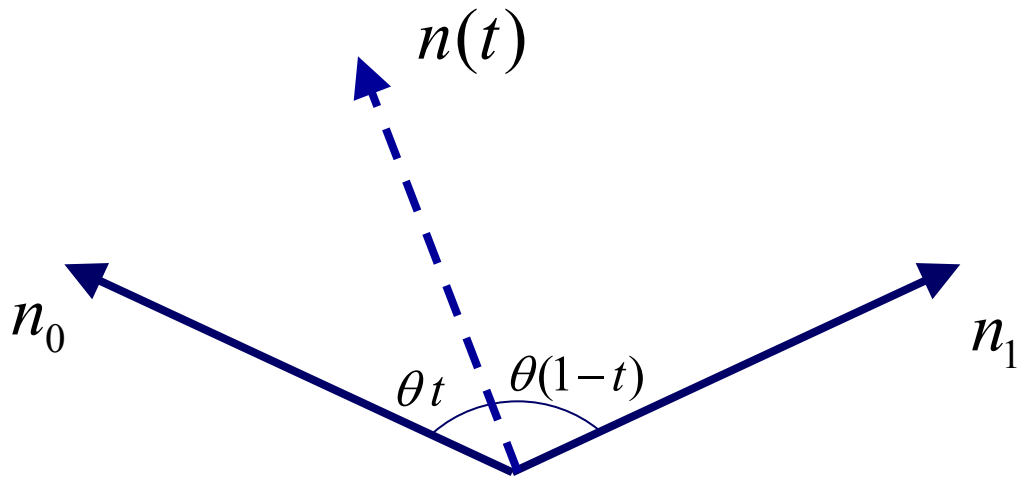
SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$

$$n_0 \times n(t) = n_0 \times (\alpha n_0 + \beta n_1)$$

$$n_1 \times n(t) = n_1 \times (\alpha n_0 + \beta n_1)$$



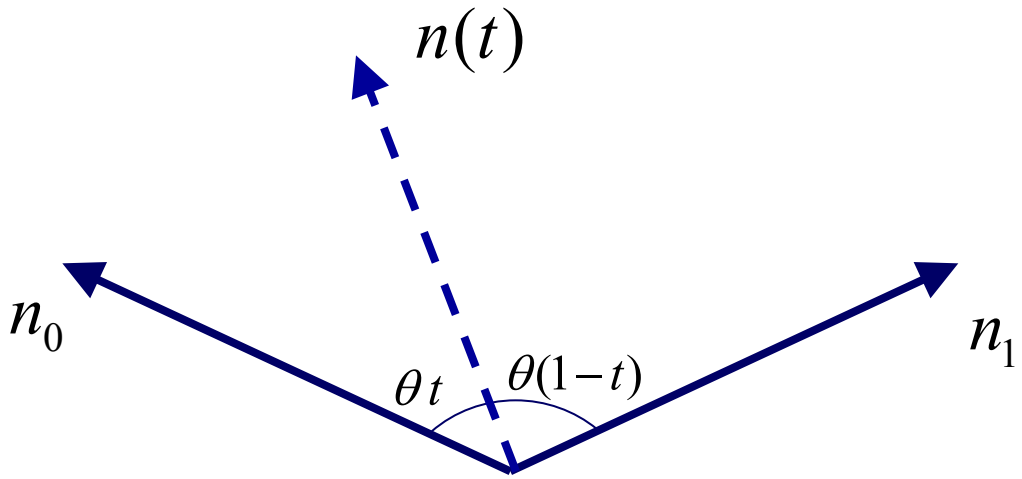
SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$

$$n_0 \times n(t) = \beta(n_0 \times n_1)$$

$$n_1 \times n(t) = \alpha(n_1 \times n_0)$$



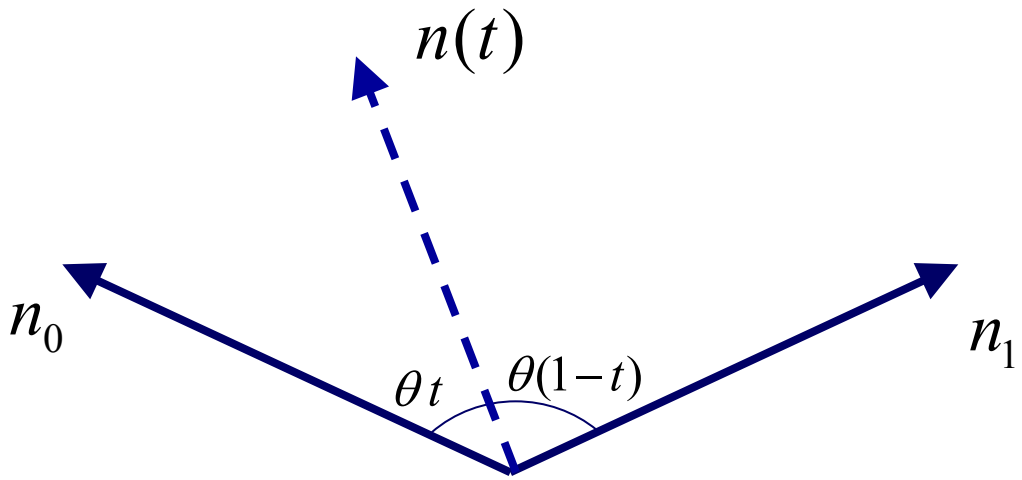
SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$

$$|n_0 \times n(t)| = \beta |n_0 \times n_1|$$

$$|n_1 \times n(t)| = \alpha |n_1 \times n_0|$$



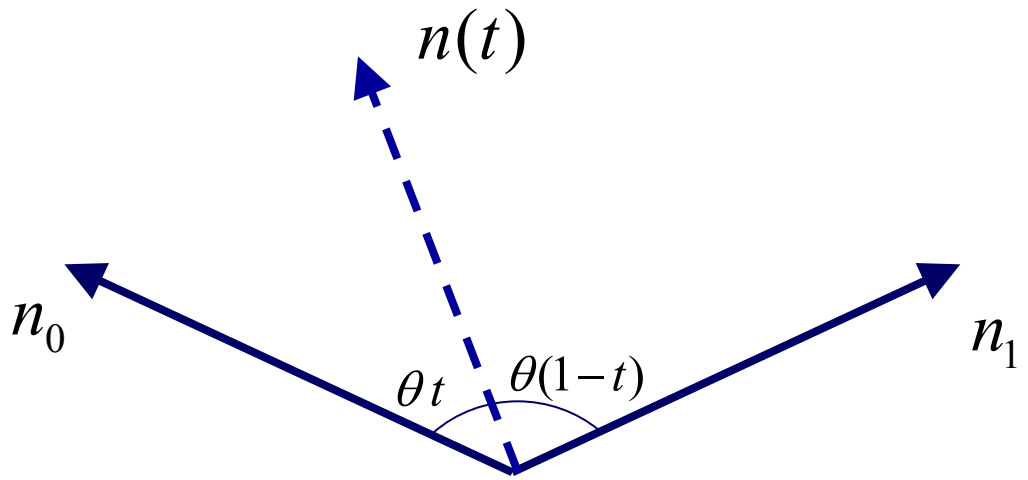
SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$

$$|n_0||n(t)|\sin(\theta t) = \beta |n_0||n_1|\sin(\theta)$$

$$|n_1||n(t)|\sin(\theta(1-t)) = \alpha |n_1||n_0|\sin(\theta)$$



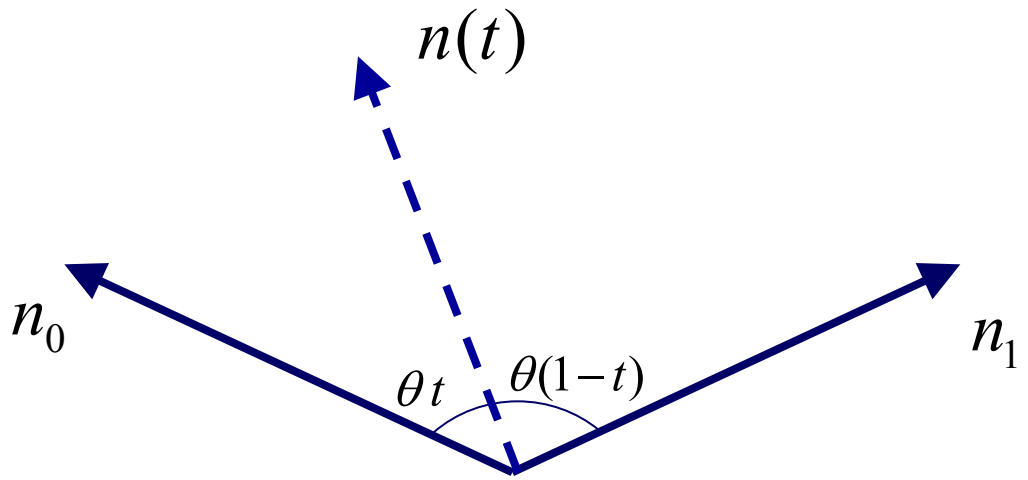
SLERP
(Spherical Linear Interpolation)

$$|n_0| = |n_1| = |n(t)|$$

$$n(t) = \alpha n_0 + \beta n_1$$

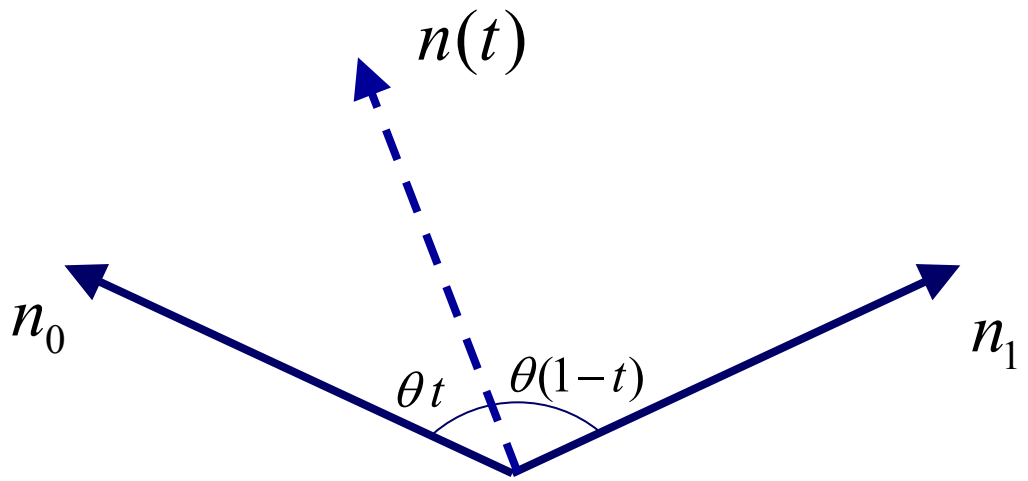
$$\sin(\theta t) = \beta \sin(\theta)$$

$$\sin(\theta(1-t)) = \alpha \sin(\theta)$$



SLERP
(Spherical Linear Interpolation)

$$n(t) = \frac{\sin(\theta(1-t))n_0 + \sin(\theta t)n_1}{\sin(\theta)}$$

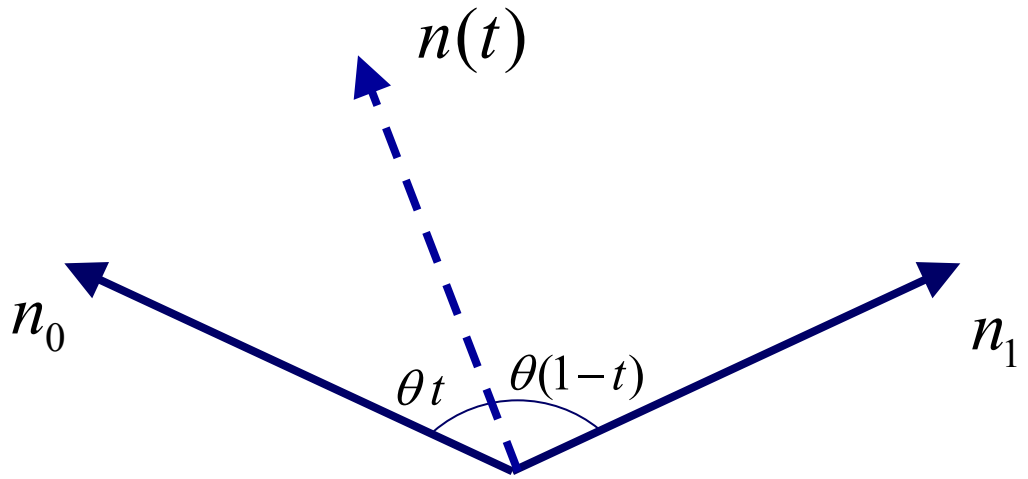


SLERP
(Spherical Linear Interpolation)

$$n(t) = \frac{\sin(\theta(1-t))n_0 + \sin(\theta t)n_1}{\sin(\theta)}$$

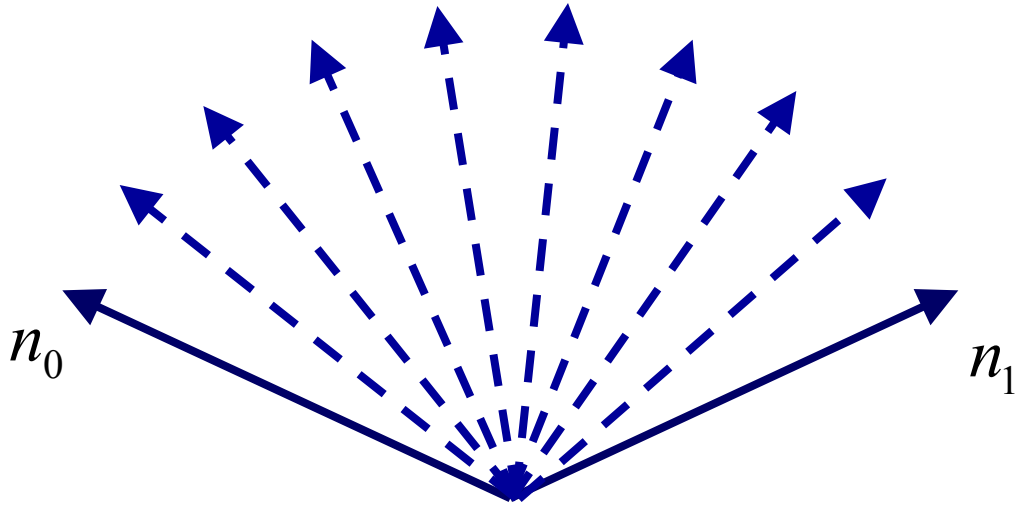
$$n(0) = n_0$$

$$n(1) = n_1$$



SLERP
(Spherical Linear Interpolation)

$$n(t) = \frac{\sin(\theta(1-t))n_0 + \sin(\theta t)n_1}{\sin(\theta)}$$



Illuminazione e Shader (GLSL)

Come si è detto, ad ogni vertice viene associato un colore e ogni pixel/fragment interno ad un triangolo ha un colore dato dall'interpolazione del colore dei vertici; ma se vogliamo applicare l'illuminazione …

…allora il colore di ogni pixel/fragment dipende da:

- Posizione e proprietà delle sorgenti luminose (luci)
- Proprietà del materiale di cui è composto l'oggetto
- Dal modello di illuminazione (modello di Phong o sua variante che comunque necessita delle normali)

Nota: la posizione della/e luci è soggetta a trasformazioni come tutte le altre primitive geometriche.

Che tipi di Luce?

- Direzionale: localizzata all'infinito (un vettore/direzione)
  v=[x,y,z,w] con w=0
- Puntiforme: v=[x,y,z,w] con w=1; irradia in tutte le direzioni
- Spotlight: concentra la luce in un cono (posizione, direzione, angolo ed esponente)
- Colore: color=[r,g,b,a]

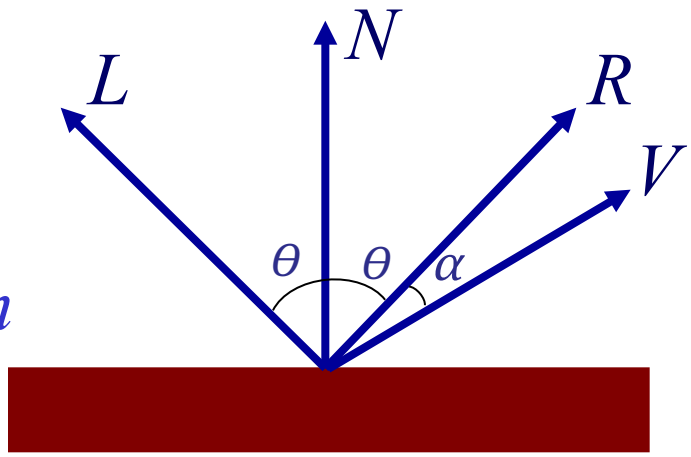
Colore Generato

Riprendiamo il Modello di Phong:

$$I = k_a I_a + I_l \left(k_d (L \cdot N) + k_s (R \cdot V)^n \right)$$

Il colore prodotto, illuminando un vertice, può essere calcolato come:

VertexColor = Ka * Ia +
Kd * Id * cos(theta) +
Ks * Is * cos(alpha)^n



GLSL (OpenGL ES Shading Language)

Per utilizzare un modello di illuminazione in WebGL si deve scrivere un opportuno programma shader in GLSL; prima di vedere qualche esempio, riprendiamo e approfondiamo alcuni concetti del linguaggio ES GLSL:

-Linguaggio ad alto livello come il C

-Nuovi titpi di dati: Matrici, Vettori, Campioni

-gli stati OpenGL ES sono resi disponibili mediante variabili predefinite

GLSL Tipi di Dati

Tipi Scalari: float, int , bool

Tipi Vectori : vec2, vec3, vec4
ivec2, ivec3, ivec4
bvec2, bvec3, bvec4

Tipi Matrice: mat2, mat3, mat4

Campionamento Texture: sampler2D, samplerCube

GLSL Costruttori

Stile C++ dei Costruttori:

```
vec2 v = vec2(1.0, 2.0);      // composto da valori
v = vec2(3.0, 4.0);           // non deve essere inizializzato
vec4 u = vec4(0.0);           // inizializza tutti gli elementi a 0
vec2 t = vec2(u);             // prende le prime due componenti
vec3 vc = vec3(v, 1.0);       // compone vec2 e float
mat2 m = mat2(1.0, 2.0,       // prima colonna
              3.0 , 4.0);     // seconda colonna
```

GLSL Operatori

Stile C/C++ degli Operatori aritmetici e logici:

Operatori overloaded per operazioni matriciali e vettoriali

```
mat4 m;
vec4 a, b, c;

b = a*m;    // un vettore riga memorizzato come un
            //array 1d
c = m*a;    // un vettore colonna memorizzato come un
            //array 1d
```

GLSL Funzioni di Libreria

-Funzioni Standard del C

-Trigonometriche : sin, cos , tan, asin , acos , atan

-Aritmetiche : pow , log2, sqrt , max, abs, min

-Funzioni su vettori:

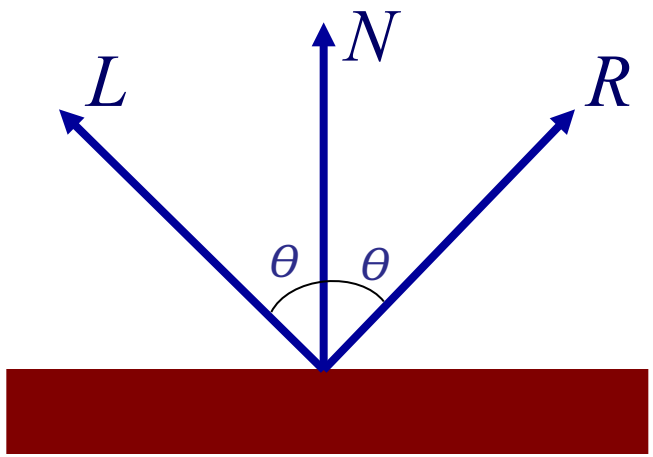
-normalize

-reflect

-length

-dot

-distance




```
R = reflect(-L, N)
```

GLSL Swizzling e Selezione

-Accesso alle componenti di vettori [ ] usando l'operatore ".":

x, y, z, w

r, g, b, a

s, t, p, q

```
vec4 a;
a[2] == a.b == a.z == a.p
```

-l'operatore Swizzling permette di selezionare componenti multiple da tipi vettore:

```
vec4 a;
a.yz = vec2(1.0, 2.0 );
a.xy = a.yx ;    // scambia gli elementi
```

GLSL Istruzioni di Controllo

if

if else

expression ? true - expression : false - expression

while, do while

for

la ricorsione è proibita

Gouraud Shading

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;
uniform int mode;   // Rendering mode
uniform float Ka;   // Ambient reflection coefficient
uniform float Kd;   // Diffuse reflection coefficient
uniform float Ks;   // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position
varying vec4 color; //color
```

continua

Gouraud Shading

```
void main(){        //vertex-shader
   vec4 vertPos4 = modelview * vec4(position, 1.0);
   vertPos = vec3(vertPos4) / vertPos4.w;
   normalInterp = vec3(normalMat * vec4(normal, 0.0));
   gl_Position = projection * vertPos4;
   vec3 N = normalize(normalInterp);
   vec3 L = normalize(lightPos - vertPos);
// Lambert's cosine law
   float lambertian = max(dot(N, L), 0.0);
   float specular = 0.0;
   if(lambertian > 0.0) {
       vec3 R = reflect(-L, N);      // Reflected light vector
       vec3 V = normalize(-vertPos); // Vector to viewer
// Compute the specular term
       float specAngle = max(dot(R, V), 0.0);
       specular = pow(specAngle, shininessVal);
      }
```

continua

Gouraud Shading

```
    color = vec4(Ka * ambientColor +
                         Kd * lambertian * diffuseColor +
                         Ks * specular * specularColor, 1.0);
// only ambient
    if(mode == 2) color = vec4(Ka * ambientColor, 1.0);
// only diffuse
    if(mode == 3) color = vec4(Kd * lambertian * diffuseColor, 1.0);
// only specular
    if(mode == 4) color = vec4(Ks * specular * specularColor, 1.0);
}

precision mediump float;
varying vec4 color;
void main( ) {.        //fragment-shader
     gl_FragColor = color;
}
```

Phong Shading

```
attribute vec3 position;
attribute vec3 normal;
uniform mat4 projection, modelview, normalMat;
varying vec3 normalInterp;
varying vec3 vertPos;

void main(){   //vertex-shader
    vec4 vertPos4 = modelview * vec4(position, 1.0);
    vertPos = vec3(vertPos4) / vertPos4.w;
    normalInterp = vec3(normalMat * vec4(normal, 0.0));
    gl_Position = projection * vertPos4;
}
```

continua

Phong Shading

```
precision mediump float;
varying vec3 normalInterp;  // Surface normal
varying vec3 vertPos;       // Vertex position
uniform int mode;   // Rendering mode
uniform float Ka;   // Ambient reflection coefficient
uniform float Kd;   // Diffuse reflection coefficient
uniform float Ks;   // Specular reflection coefficient
uniform float shininessVal; // Shininess
// Material color
uniform vec3 ambientColor;
uniform vec3 diffuseColor;
uniform vec3 specularColor;
uniform vec3 lightPos; // Light position
```

continua

Phong Shading

```
void main() {   //fragment shader
    vec3 N = normalize(normalInterp);
    vec3 L = normalize(lightPos - vertPos);
// Lambert's cosine law
   float lambertian = max(dot(N, L), 0.0);
   float specular = 0.0;
   if(lambertian > 0.0) {
        vec3 R = reflect(-L, N);      // Reflected light vector
        vec3 V = normalize(-vertPos); // Vector to viewer
// Compute the specular term
        float specAngle = max(dot(R, V), 0.0);
        specular = pow(specAngle, shininessVal);
    }
    gl_FragColor = vec4(Ka * ambientColor +
                        Kd * lambertian * diffuseColor +
                        Ks * specular * specularColor, 1.0);
```

continua

Phong Shading

```
// only ambient
  if(mode == 2) gl_FragColor = vec4(Ka * ambientColor, 1.0);
// only diffuse
  if(mode == 3) gl_FragColor = vec4(Kd * lambertian * diffuseColor, 1.0);
// only specular
  if(mode == 4) gl_FragColor = vec4(Ks * specular * specularColor, 1.0);
}
```

Demo

http://www.cs.toronto.edu/~jacobson/phong-demo/

Esercizi

In HTML5_webgl_2 analizzare e sperimentare i codici:

cube_shading.html e .js
F3d_lighting_point.html e .js
F3d_lighting_point_color_UI.html e .js
F3d_lighting_point_specular_power_UI.html e .js
F3d_lighting_point_specular_power_GUI.html e .js

Analizzare con attenzione i programmi vertex-shader e fragment-shader che implementano differenti varianti di illuminazione

Illuminazione e Texture

Se simuliamo l'illuminazione della scena e si vuole texturare un oggetto quali attributi o parametri devono essere modificati per produrre un effetto realistico?

I parametri più comunemente usati per il texture mapping sono i coefficienti di illuminazione ambiente e di riflessione diffusa (o colore) dell'oggetto, cioè Ka e Kd.

Ancora si può agire sulla normale una cui perturbazione influenza la riflessione della luce producendo un effetto simile a quello di un materiale non liscio.

Esercizi

In HTML5_webgl_2 analizzare e sperimentare il codice:

cube_texture_mtl.html e .js

Analizzare con attenzione i programmi vertex-shader e fragment-shader che implementano il modello di illuminazione di phong insieme all'applicazione di una texture.

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Giulio Casciola

Dip. di Matematica

giulio.casciola at unibo.it