Graphics Programming

Master Computer Game Development 2013/2014

Illuminazione e Shading

University of Verona

luca.giona@3dflow.net

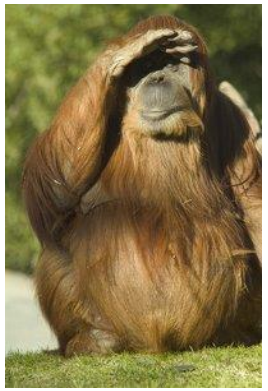
Modello di illuminazione

- Come vediamo i nostri oggetti se non ci sono luci?
- Dobbiamo cercare di simulare la luce e modellare l'interazione con ogni superficie della scena se vogliamo creare effetti realistici.

Definizioni

- Prima di addentrarci nel vedere quali sono i modelli di illuminazione adatti/utilizzati nel rendering real-time diamo qualche definizione.
- Si fa spesso confusione tra
  - Lighting (Illuminazione)
  - Shading (tonalità - ombreggiatura)

    I significati letterali dei termini non sono di molto aiuto…



Illuminazione e Shading

- Lighting
  - Il processo che calcola l'intensità luminosa (ovvero la luce trasmessa) di un punto tridimensionale.
- Shading
  - Il processo che assegna i colori ai pixel.
- "L'intensità luminosa" viene di solito calcolata in modo indipendente per le tre componenti di colore R-G-B

Definizioni

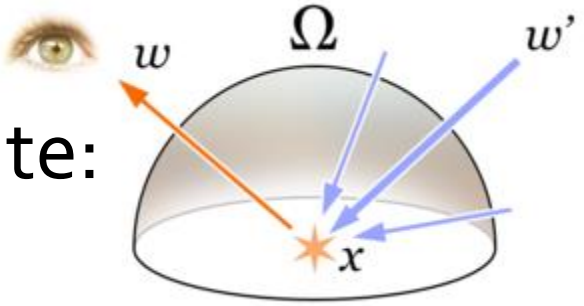
- L'energia elettromagnetica interagisce con gli oggetti della scena; ciò che noi vediamo, la luce, è energia elettromagnetica nello spettro della luce visibile che colpisce gli occhi.
- La luce di una scena dipende da numerosi fattori:
  - Proprietà delle sorgenti luminose.
  - Proprietà dei materiali
  - Posizione degli oggetti e delle sorgenti luminose.

Modelli di illuminazione

- Esistono vari modelli di illuminazione.
- Fondamentalmente possono essere distinti tra locali/empirici e globali/fisici.
- Empirici e locali
  - Soluzioni empiriche per ottenere in modo veloce risultati "realistici".
  - Phong - Blinn(locale).
- Modelli globali
  - Cercano di risolvere (con assunzioni semplificanti) l'equazione fondamentale del rendering
  - Raytracing , Radiosity, Photon Mapping, Path Tracing

Equazione fondamentale del rendering

- La "realtà fisica" sommariamente:



$$L_o(x,\vec{w}) = L_e(x,\vec{w}) + \int_\Omega f_r(x,\vec{w}',\vec{w}) L_i(x,\vec{w}')(\vec{w}' \cdot \vec{n}) d\vec{w}'$$

- Lo : quantità di luce diretta in direzione w a partire da un punto x.
- Le : luce emessa dal punto x in direzione w + termine quantità di luce riflessa.
- w' direzione luce incidente.
- Li: quantità di luce incidendente con direzione w'
- Fr : BRDF, rapporto di luce incidente/riflessa
- (w' n): attenuazione luce dovuta all'angolo di incidenza.

Equazione fondamentale del rendering.

- Difficile (impossibile) da risolvere in tempi ragionevoli.
  - Si devono considerare le interazioni oggetto-oggetto tra la luce (metodo globale).
  - La BRDF è difficile da misurare per superfici reali.
  - E noi abbiamo anche il vincolo real time!

Radiosity (idea)

- Algoritmo Radiosity:
  - Cerca di risolvere l'equazione del rendering assumendo la scena composta da soli diffusori perfetti (superfici lambertiane)
  - L'illuminazione di una patch è ottenuta dalla luce emessa (se sorgente) e dalla luce che arriva dalle altre patch.
  - Illuminazione View-Independent
  - Si devono calcolare i form factor ovvero la percentuale di luce che una superficie riflette su un altra.
  - Step: sistema lineare con tutti i form factors.

Radiosity Form Factors

The "full matrix" radiosity solution requires form factors between each surface to be calculated, and the following equation to be solved:

$$\begin{bmatrix} 1-\rho_1 F_{11} & -\rho_1 F_{12} & \cdots & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1-\rho_2 F_{22} & \cdots & -\rho_2 F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \cdots & 1-\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{bmatrix}$$

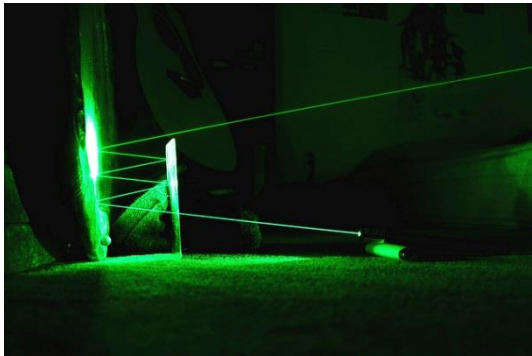
ρ_i is the reflectivity of surface i,
F_{ij} is the form factor from surface i to surface j,
B_i is the radiosity of surface i, and
E_i is the emission of surface i.

Radiosity

- Troppo oneroso per il real-time (esistono approssimazioni).

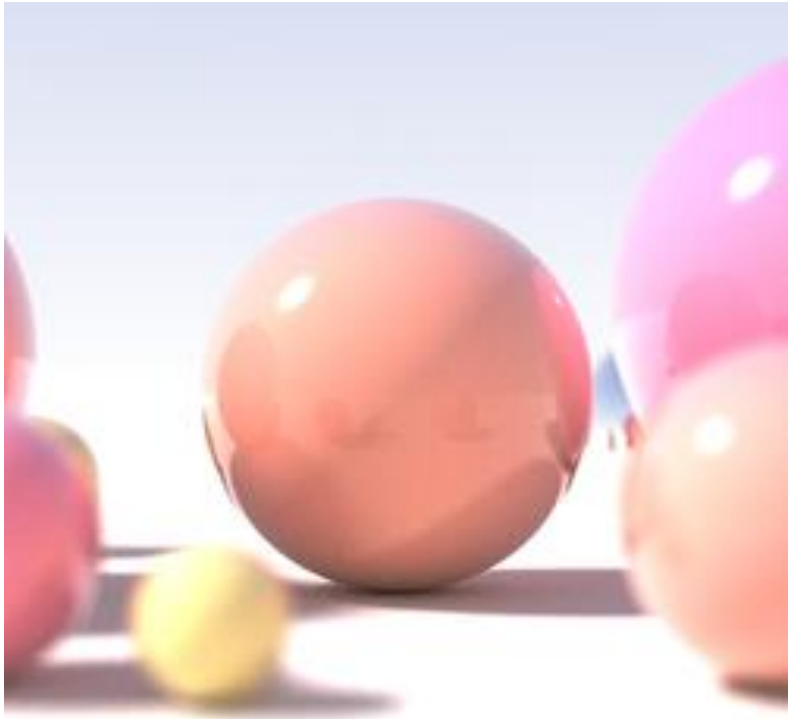
Ray Tracing

- Complementare di radiosity: calcola illuminazione assumendo superfici speculari.
  - Idea: calcola il percorso inverso dei raggi che arrivano al nostro occhio.
    - Quando i raggi incontrano la superficie, saranno riflessi su un'altra.
    - Fai il percorso inverso dei raggi fino a trovare la sorgente luminosa.
    - Metodo ricorsivo.
    - View Dependent



Ray Tracing

- Troppo oneroso per real time (esistono approssimazioni)



Altre tecniche.

- Radiosity e Ray Tracing possono venire utilizzati assieme.
- Altri metodi:
  - Pathtracing: estensione ray tracing per ottenere illuminazione superfici lambertiane (si lanciano campioni di raggi a caso – metodo di Monte Carlo)
  - Photon Mapping: estensione del ray tracing, si traccia non solo il percorso inverso ma anche il percorso diretto della luce.
- Nessuno di questi è veramente utilizzabile in real time (anche se esistono versioni approssimate in grado di funzionare su semplici scene)!

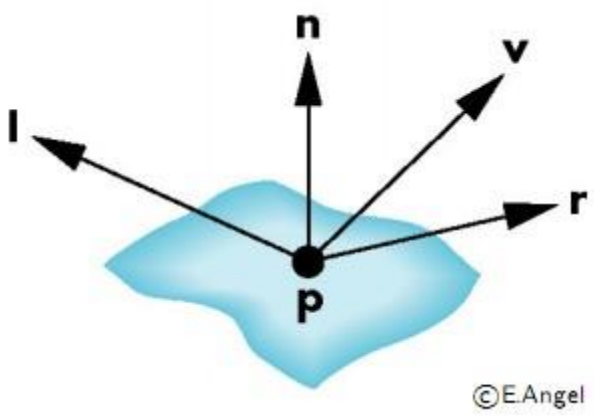
Modello di illuminazione di Phong

- Nell'illuminazione real-time si usa di solito un metodo di illuminazione locale e assolutamente empirico: il modello di phong.
- L'intensità luminosa (o luce) di un punto viene scomposta in 4 componenti: **Ambientale, Diffusiva, Speculare e Emissiva**

  $\mathbf{I} = \mathbf{I}amb + \mathbf{I}diff + \mathbf{I}spec + \mathbf{I}emiss$

Modello di illuminazione di Phong

- In un modello locale non vi sono ombre e conta solo la direzione sotto cui il punto vede la luce.

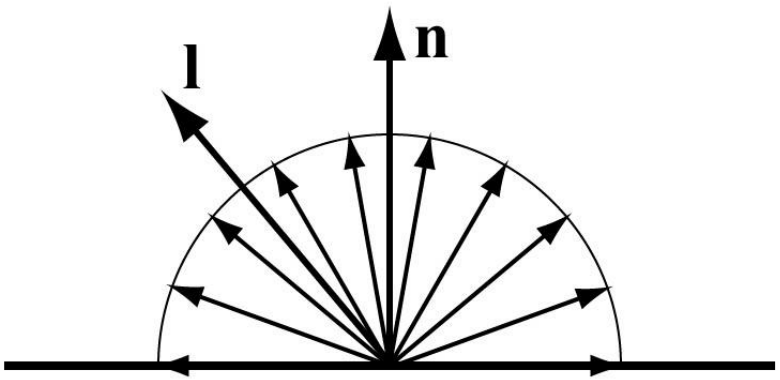


Componente Ambientale

- I modelli locali non simulano le interriflessioni tra i vari oggetti.
- Si introduce quindi un termine di luce ambientale che tiene conto "indirettamente" delle interriflessioni
- $I_{a\,out} = I_a K_a$
- K_a è chiamato coefficiente (o un vettore di coefficienti RGB) di riflessione ambientale, compreso tra 0 e 1.

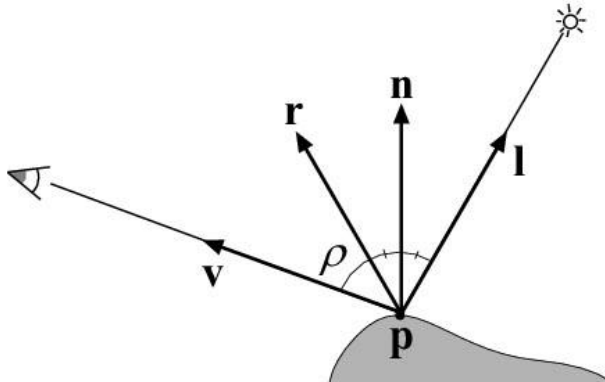
Componente diffusiva

- La componente diffusiva deriva direttamente dalla legge di Lambert: i fotoni sono dispersi uniformemente in tutte le direzioni $i_{diff} = \mathbf{n} \cdot \mathbf{l} = \cos\phi$
- Dipende dalla direzione della luce incidente e dalla normale alla superficie.
- $I_{dout} = Ik_d \cos\theta = Ik_d(n \cdot l)$
- K_d coefficiente di Riflessione diffusiva



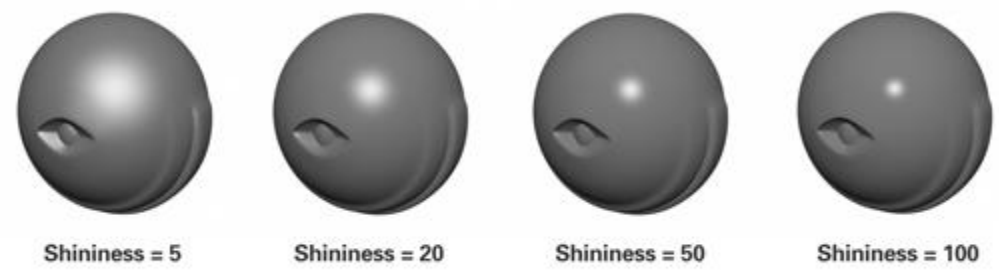
Componente Speculare (di Phong)

- In una superficie riflettente, la luce viene riflessa in una direzione r dipendente dalla direzione di incidenza l e dalla normale alla superficie n.
- Si dimostra che il vettore r è: $\mathbf{r} = -\mathbf{l} + 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n}$
- Phong introduce il seguente modello di riflessione speculare: $I_{sout} = Ik_s(r \cdot v)^n$
- K_s è il coefficiente di riflessione speculare, mentre n è l'esponente di riflessione speculare.



Componente Speculare (di Phong)

- La componente speculare di phong non ha nessun significato fisico.
- N modula la "lucidità" di una superficie. Per n → infinito si ha una riflessione speculare perfetta.



Componente emissiva

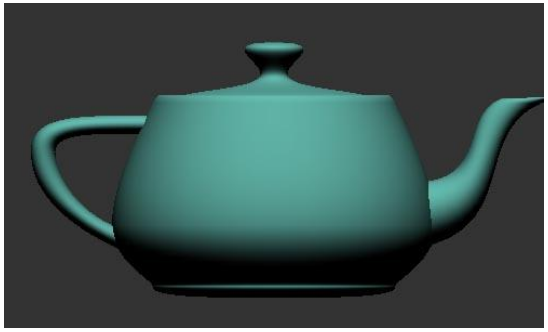
- Può essere che il materiale dell'oggetto emetta luce propria.
- Si può aggiungere quindi un'intensità di emissione.
- $I_{eout} = K_e$
- La Componente emissiva non crea una sorgente luminosa ma è solo locale.

Sommando…

Ambientale

+

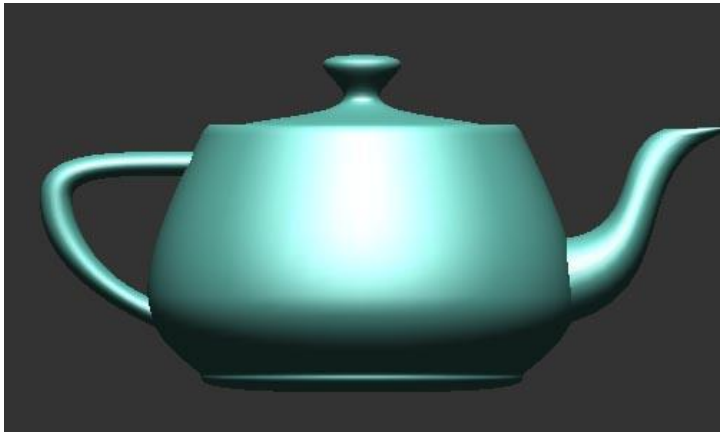
Diffusiva



+

Speculare

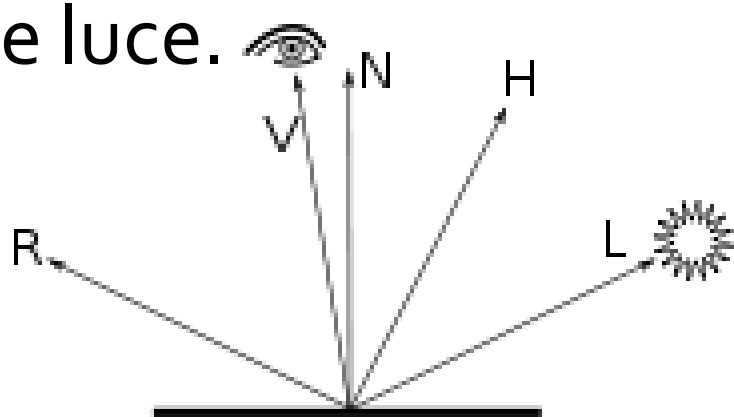
=



Modello di Blinn-Phong

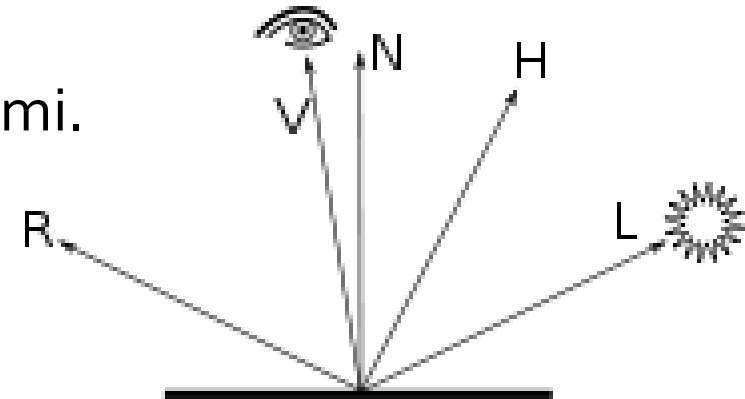
- Un modello alternativo per calcolare la componente speculare è stato introdotto da Blinn e prende il nome di modello di Blinn-Phong.
- Possiamo calcolare l'halfangle vector, ovvero il vettore unitario che sta in mezzo tra la normale e il vettore direzione luce.

$$H = \frac{L + V}{|L + V|}$$



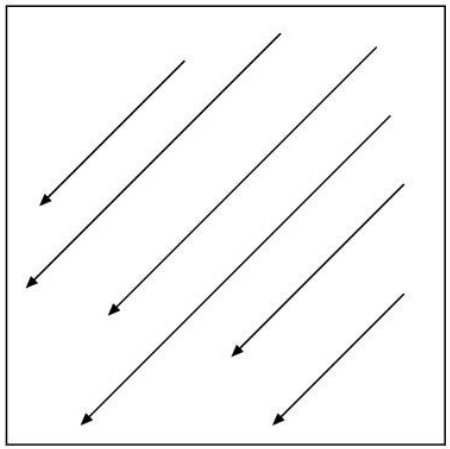
Modello di Blinn-Phong

- A questo punto la componente speculare diventa:
  - $I_{sout} = I\, k_s\, (n \cdot h)^n$
  - $(n \cdot h)^n$ prende il posto di $(r \cdot v)^n$
  - L'angolo corrispondente al dot product $n \cdot h$ è chiamato halfway angle ed è più piccolo di quello tra r e v.
  - Possiamo ottenere effetti simili variando l'esponente n.
  - E' stato dimostrato che produce risultati più simili a BRDF reali
  - Più efficiente con sorgenti puntiformi.

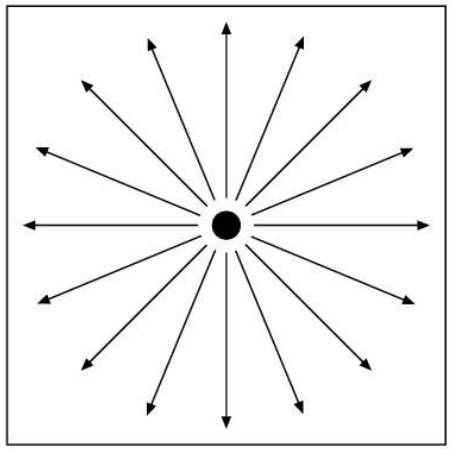


Tipi di sorgenti luminose

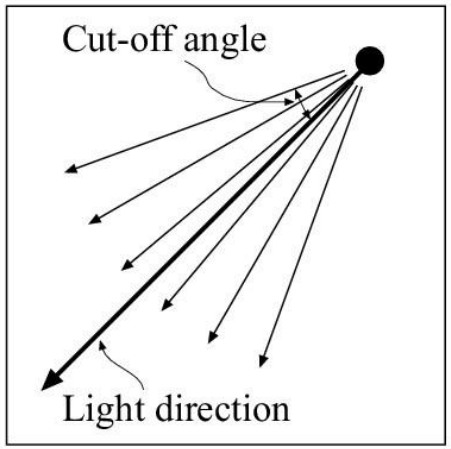
- Possiamo avere diverse sorgenti luminose, basta sommare i loro contributi.
- Possiamo avere diversi tipi di sorgenti luminose.
  - Le luci direzionali(come il sole) sono poste a distanza infinita con una direzione di incidenza sempre uguale.
  - Nelle point light- e spot light la direzione di incidenza è data dalla differenza tra la posizione del punto e la posizione della luce



Directional Light



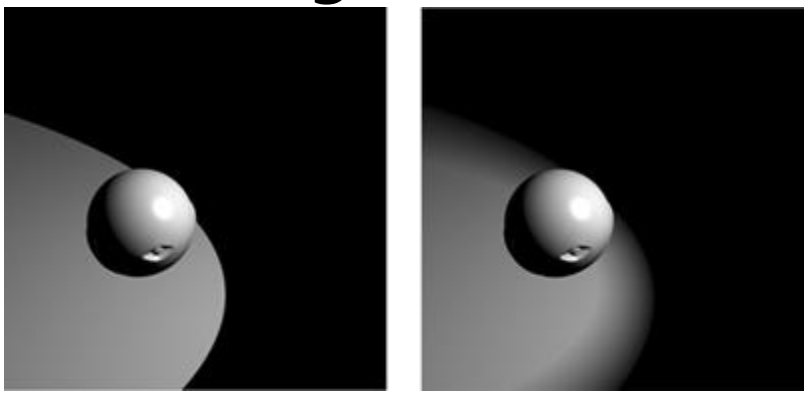
Point Light

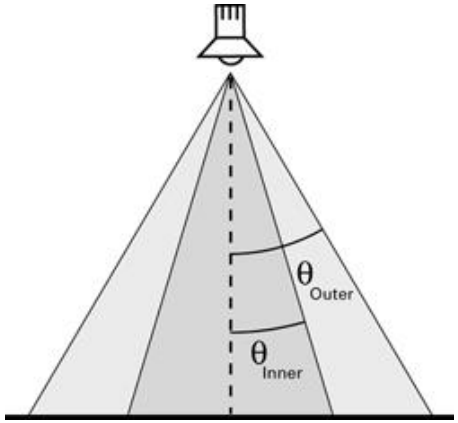


Spot Light

Spotlight

- Le luci “spotlight” generano luce solo entro un certo cono.
- Eventualmente si definiscono un angolo interno in qui la luce è piena e un angolo esterno in cui la luce è attenuata in base all’angolo.





Attenuazione luce con distanza

- Spesso si utilizza una attenuazione della luce con la distanza per le componenti diffusive e speculari.
- Kc kl e kq sono costanti che determinano il rapporto di attenuazione in base alla distanza.

$$attenuationFactor = \frac{1}{k_C + k_L d + k_Q d^2}$$

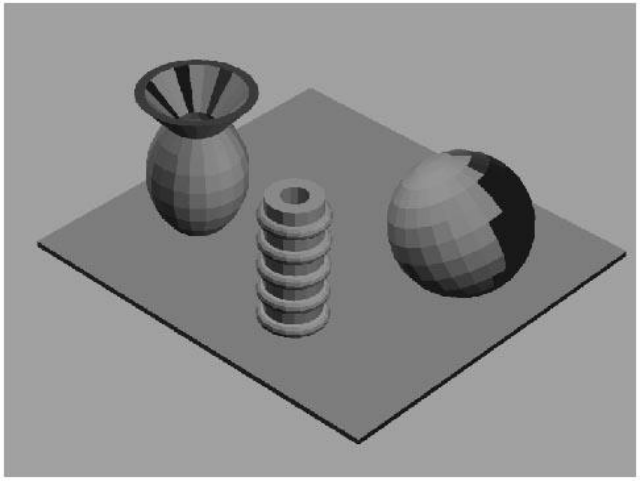
lighting = *emissive* + *ambient* + *attenuationFactor* x (*diffuse* + *specular*)

Shading

- Lo shading è deputato a determinare il colore effettivo dei pixel a partire da un modello di illuminazione dato.
- Determina come e quando applicare il modello di illuminazione prescelto.
- Vi sono tre tipi di shading: Flat, Gourad e Phong.

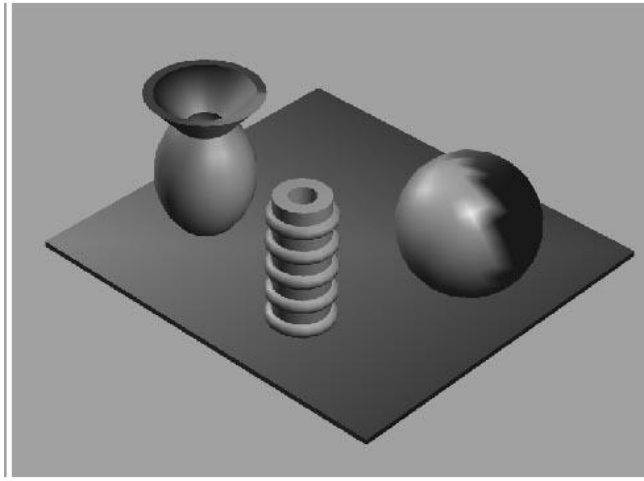
Flat shading

- A ogni primitiva geometrica (e.g. Triangoli) è associato uno stesso colore uniforme.
- Il più semplice, il più veloce e il più brutto.



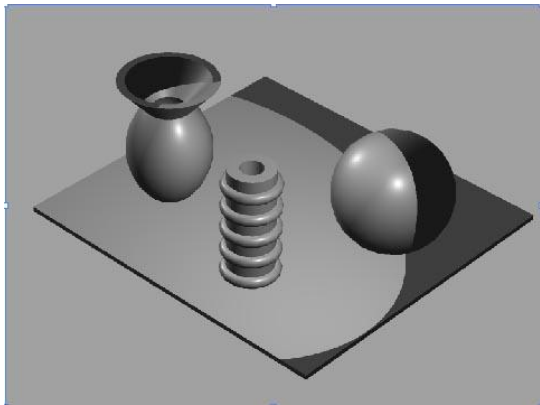
Gourad shading

- Shading interpolativo: il valore di illuminazione viene calcolato ai vertici e viene interpolato per i pixel (fragment) interni.
- Risultati migliori rispetto al flat. Utilizzato di default ad esempio dalla fixed pipeline di OpenGL e DX9.



Phong shading

- Vengono interpolate le normali ai vertici per ottenere delle normali dei pixel(fragment) interni.
- Il modello di illuminazione viene calcolato per pixel(fragment).
- E' il più oneroso e bisogna tenere le normali e i vettori di incidenza della luce.
- I risultati però sono i migliori.



Illuminazione e shading in DirectX 10/11

- DirectX 10/11 ha eliminato le parti fixed di illuminazione.
- Non esistono più le chiamate a funzione per impostare luci e proprietà dei materiale.
- Dovremo essere noi a specificare il modello di illuminazione e di shading nei nostri vertex e pixel shader.

Esempio illuminazione e shading

- Proviamo ora a implementare un esempio che utilizzi il modello di illuminazione di phong e che utilizzi gourad-shading o phong-shading.
- La scena sarà composta da un cubo, un piano e due luci puntiformi che ruotano attorno ad essi.
- Per passare da un modello di illuminazione all'altro basterà premere il pulsante spazio.

Definizione variabili

- Prima di addentrarci nel cuore dell'esempio, definiamo tutte le variabili che entrano in gioco.
- Dovremo innanzitutto definire delle strutture luce e materiale sia nel sorgente del programma sia negli shaders.
- Utilizzeremo una struttura per descrivere le proprietà del materiale e una per descrivere tutte le luci.

Definizioni strutture

- Struttura per un materiale associato ad una mesh:

```
struct Material
{
        DirectX::XMFLOAT4 ambiental;
        DirectX::XMFLOAT4 diffusive;
        DirectX::XMFLOAT4 specular;
        DirectX::XMFLOAT4 emissive;
        float shininess;
        float padding[3];
};
```

Definizioni strutture

- Struttura materiale nell'effetto:

```
cbuffer Material : register(cbo)
{
        float4 Ambiental;
        float4 Diffusive;
        float4 Specular;
        float4 Emissive;
        float Shininess;
};
```

Definizioni strutture

- Definiamo quindi anche una struttura per le luci (fino a 5 luci):

```
const unsigned int MaxLights = 5;
struct Lights
{
        DirectX::XMFLOAT4 position[MaxLights];
        DirectX::XMFLOAT4 color[MaxLights];
        DirectX::XMFLOAT4 ambient[MaxLights];
        DirectX::XMFLOAT4 attenuation[MaxLights];
        int usedLights; // quante luci sono utilizzate.
        float padding[3];
};
```

Definizioni strutture

- La relativa struttura nello shader sarà:

```
#define MaxLights 5
cbuffer LightsBuffer : register(cb1)
{
        float4 LightPosition[MaxLights];
        float4 LightColor[MaxLights];
        float4 LightAmbient[MaxLights];
        float4 LightAttenuation[MaxLights];
        int UsedLights;
};
```

Collegamento variabili sorgente/shaders

- Prima di ogni draw call dovremo collegare le variabili del materiale con quelle del materiale corrente.
- La stessa cosa sarà effettuata per le luci:

```
pd3dDeviceContext->PSSetConstantBuffers (0, 1, &materialCBuffer);
pd3dDeviceContext->PSSetConstantBuffers (1, 1, &lightsCBuffer);
```

Inizializzazione parametri

- Inizializzate i parametri del materiale e delle luci a piacimento. Ricordandovi che è un modello additivo:
  - La componente ambientale di solito è settata su a valori bassi
  - La componente diffusiva mi determina il colore effettivo dell'oggetto.
  - La componente speculare determina il colore delle riflessioni speculari (highlights).
  - La componente emissiva raramente è utilizzata (i.e. In un modello additivio viene settata a nero).
  - I colori delle luci a piacimento

Shaders

- Dovremo definire due tecniche con due shader differenti, un VS e PS per il Gourad-Shading e un VS e PS per il Phong-Shading

Vertex Shader input/output

- Vediamo prima gli shaders per il Gourad-Shading.
- Il Vertex shader dovrà prendere in input anche normali e dovrà mandare in output posizione e colori che arriveranno al Pixel Shader.

Vertex Shader Input/Output

```
// Strutture Input e Output per i Vertex shaders
struct VS_INPUT
{
        float4 pos      : POSITION;     //posizione
        float3 norm     : NORMAL;       //normale
};

struct VS_OUTPUT
{
        float4 pos              : SV_POSITION;
        float4 color            : COLOR0;
};
```

Vertex Shader - Gourad

```
PixelShaderInput GouradShadingVS(VertexShaderInput input)
{
   PixelShaderInput output;

   float4 worldSpacePos = mul(World, float4(input.pos, 1.0f));
   float4 cameraSpacePos = mul(View, worldSpacePos);
   float3 worldSpaceNormal = mul((float3x3)World, input.norm);
   float3 viewDir = normalize(CameraPosition - worldSpacePos.xyz);

   output.pos = mul(Projection, cameraSpacePos);
   output.col = CalcLightinig(worldSpacePos.xyz, worldSpaceNormal, viewDir);
   return output;
}
```

Vertex Shader - Gourad

- Il vettore direzione punto-camera è calcolato utilizzando CameraPosition, che sarà settata dal programma e che conterrà il punto eye (posizione telecamera).

```
float3 viewDir = normalize(CameraPosition - worldSpacePos.xyz);
```

Vertex Shader - Gourad

- Le normali vengono trasformate assieme ai vertici in coordinate mondo.
- Il colore viene effettivamente calcolato nella funzione CalcLighting a cui viene passato:
  - La posizione (in coordinate mondo) del vertice
  - La normale nel vertice
  - Il vettore direzione punto-camera normalizzato

Calcolo illuminazione

- Il calcolo dell'illuminazione viene effettuato nella funzione CalcLighting.
- Il valore del colore finale viene memorizzato nella variabile color.

```
float4 CalcLightinig(float3 worldPosition, float3 worldSpaceNormal,
                                float3 viewDirection)
{
   float3 color = float3(0.0f, 0.0f, 0.0f);
   // Aggiungi componenti modello illuminazione phong tutte le luci.....
   .......
}
```

CalcLighting

- All'interno di calcLighting dovremo sommare le componenti per ogni sorgente luminosa:

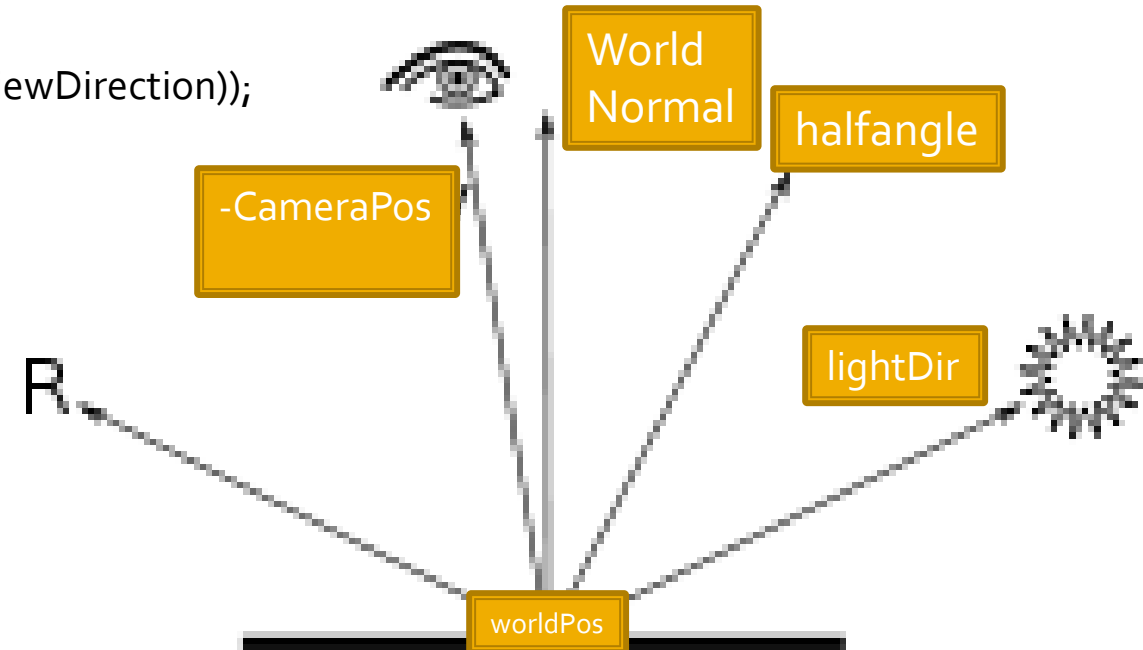
```
// colore di output
float3 color = float3(0,0,0);
 // Componente emissiva
 color += Emissive.rgb;
for(int i = 0; i < UsedLights; ++i)
{
                ....
}

return float4(color, 1.0f);
```

CalcLighting: definizione vettori

- Definiamo i vettori utili per il calcolo:

```
// Vettore luce-punto
float3 worldToLight = LightPosition[i].xyz - worldPosition;
// Distanza dalla luce
float lightDist = length( worldToLight );
// Versore luce-punto normalizzato
float3 lightDir = normalize(worldToLight);
// Vettore halfvector
float3 halfVector = normalize( (lightDir + viewDirection));
```



CalcLighting: coefficiente attenuazione

- Calcoliamo il coefficiente di attenuazione che sarà applicato alla componente diffusiva e speculare.

```
// Coefficiente di attenuazione
float fAtten = 1.of / dot( LightAttenuation[i].xyz, float3(1.of, lightDist, lightDist*lightDist) );
```

CalcLighting: componente ambientale

- Aggiungiamo la componente ambientale. Il calcolo è semplicemente:

```
color += Ambiental.rgb * LightAmbient[i].rgb;
```

CalcLighting: componente diffusiva

- Aggiungiamo la componente diffusiva:
  - max(0, Intensità*k_d(n · l) * CoefficienteAttenuazione)

- Note:
  - Il valore potrebbe essere negativo se la normale punta in direzione opposta alla luce. In questo caso non aggiungiamo nulla (clamping a 0)
  - Moltiplichiamo il valore da aggiungere per un coefficiente di attenuazione basato sulla distanza punto-luce.

CalcLighting: componente speculare

- Calcoliamo la componente riflessiva:
  - max(0,Intensità* $k_s (n \cdot h)^n$ * CoefficienteAttenuazione)

- Note:
  - Il valore potrebbe essere negativo se vediamo l'oggetto da dietro (normale direzione opposta). In questo caso non aggiungiamo nulla (clamping a 0)
  - Moltiplichiamo il valore da aggiungere per un coefficiente di attenuazione basato sulla distanza punto-luce.

CalcLighting

- All'interno di CalcLighting abbiamo calcolato il colore di un punto – in questo caso vertice – utilizzando il modello di illuminazione di Blinn-Phong.
- Il nostro PS di Gourad non dovrà fare nulla se non passare il colore automaticamente interpolato dal rasterizer nei vari fragment:

```
float4 GouradShadingPS(PixelShaderInput input) : SV_TARGET
{
   return input.col;
}
```

Phong Shading.

- Vediamo ora come ottenere uno Shading di tipo Phong.
- In questo caso il modello di illuminazione non è più applicato ai vertici ma direttamente ai fragments.
- Questo significa che dovremo passare al pixel shader le normali e le posizioni (in coordinate mondo) dei pixel.
- Normali e coordinate in output dal vertex shader mondo saranno interpolate dal rasterizer e otterremo i valori dei pixel/fragment.

Phong Shading

- L'input del vertex shader rimane invariato, mentre l'output / input del pixel shader diventa:

```
struct PixelShaderInput
{
        float4 pos: SV_POSITION;
        float3 wNormal: NORMAL;
        float3 viewDirection: VIEWDIRECTION;
        float3 wPos: WORLDPOSITION;
};
```

- Nota: passiamo normali, posizione e direzione di vista come vettori float3.
- Nello Shader Model 4 possiamo passare fino a 16 variabili in questo modo in output al vertex Shader.

Phong Shading – Vertex shader

- Il Vertex shader si occuperà di passare normali e posizioni nello spazio mondo.
- Non ci preoccupiamo del colore che sarà calcolato totalmente nel pixel shader:

```
float4 PhongShadingPS(PixelShaderInput input) : SV_TARGET
{
   return CalcLightinig(input.wPos, normalize(input.wNormal), normalize(input.viewDirection));
}
```

Pixel Shader

- Il pixel shader calcolerà il colore utilizzando CalcLighting.
- CalcLighting è la stessa funzione utilizzata nel vertex shader di Gourad.
- Ricordiamoci di rinormalizzare le normali!

Rendering

- Ad ogni frame:
  - Riaggiorniamo le matrici View e posizione telecamera, uguali per tutti gli oggetti.
  - Riaggiorniamo le posizioni delle luci, uguali per tutti gli oggetti.
  - Disegniamo gli oggetti agganciando correttamente i constant buffer con le informazioni necessarie.

Esempio 04

- Esercizio 04:
  - Provare a modificare materiali e luci.
  - Provare ad imprementare tipi differenti di luci (per esempio una spot light).

Colore non uniforme della superficie

- Nell'esempio abbiamo specificato un materiale uniforme per tutto l'oggetto.
- In precedenza avevamo visto come fosse possibile specificare in un vertex buffer il colore di ogni vertice.
- Tale colore può essere utilizzato direttamente nel calcolo dell'illuminazioni determinando i coefficiente di riflessione diffusiva (e ambientale) variabili nella superficie.

Performance shader

- Nell'esempio è stato introdotto un ciclo for dinamico sul numero di luci.
- Gli IF / else (così come switch/case, while, for condizionali) degradano le prestazioni degli shader.
  - Sono stati utilizzati solo per rendere il codice più leggibile.
- Eliminare le operazioni condizionali spesso significa scrivere più shaders o generarli/compilarli dinamicamente oppure utilizzare tecniche piu' complesse (es: deferred lighting).
  - Nei giochi AAA vi possono essere migliaia di shader differenti!