Mô hình chiếu sáng Blinn - Phong

- Hay còn gọi là mô hình chiếu sáng Phong sửa đổi
- Công thức của chiếu sáng Phong:

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{v} \cdot \mathbf{r})^{\alpha} + k_a I_a$$

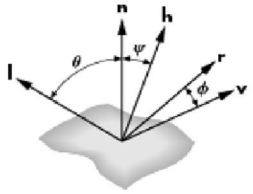
- Công thức chiếu sáng Blinn đề xuất sửa đổi

$$I = k_d I_d \, \mathbf{l} \cdot \mathbf{n} + k_s I_s (\mathbf{n} \cdot \mathbf{h})^{\beta} + k_a I_a$$

Used in OpenGL

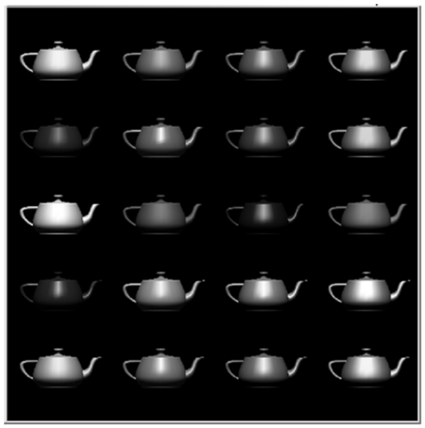
- Blinn đề xuất sử dụng vector halfway (h), hiệu quả hơn.
- h là vector đơn vị:

$$\mathbf{h} = (\mathbf{l} + \mathbf{v}) / |\mathbf{l} + \mathbf{v}|$$



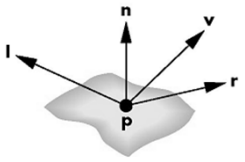
Ví dụ

- Mô hình chiếu sáng Phong sửa đổi cho kết quả tương tự như mô hình chiếu sáng Phong



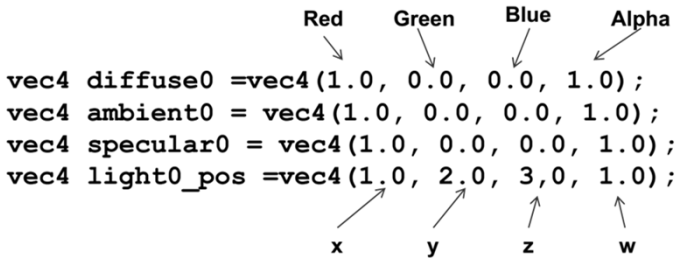
Tính toán các vector

- Để tính toán chiếu sáng tại một đỉnh P
  - Cần tính các vector l, n, r và v tại đỉnh P
- Người dùng phải đặc tả:
  - Vị trí nguồn sáng
  - Vị trí camera
  - Đỉnh (vị trí lưới)
- l = vị trí nguồn sáng – vị trí đỉnh P
- v = vị trí camera – vị trí đỉnh P
- Chuẩn hóa tất cả các vector



Đặc tả nguồn sáng điểm

- Với mỗi thành phần của nguồn sáng, thiết lập RGBA và vị trí.
- A: alpha = Độ trong suốt (0..1)– mặc định là 1: tô đặc



Khoảng cách và hướng

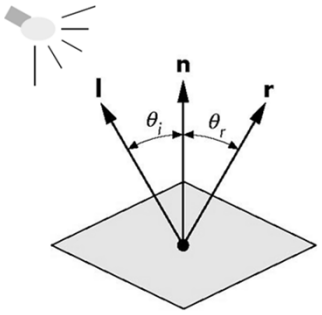
```
vec4 light0_pos =vec4(1.0, 2.0, 3,0, 1.0);
```

x y z w

- Vị trí thuộc hệ tọa độ thuần nhất

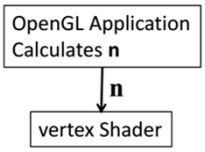
Vector phản xạ gương - r

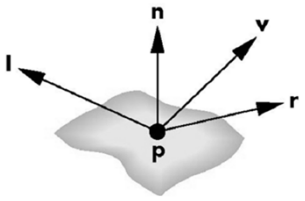
- Có thể tính r từ l và n: $r = 2(l \cdot n)n - l$
- l, n, r là đồng phẳng
- Tính vector pháp tuyến n như thế nào?



Tính vector pháp tuyến n

- Vector pháp tuyến n được tính trong ứng dụng và chuyển đến vertex shader





Phương pháp Newell cho vector pháp tuyến

- Khi tính vector pháp tuyến từ tích có hướng của 2 vector chỉ phương của bề mặt mà góc hợp bởi chúng nhỏ thì tạo ra vector pháp tuyến nhỏ →
- Công thức tính: Pháp tuyến N=(m_x, m_y, m_z)

$$m_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i + z_{next(i)})$$

$$m_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i + x_{next(i)})$$

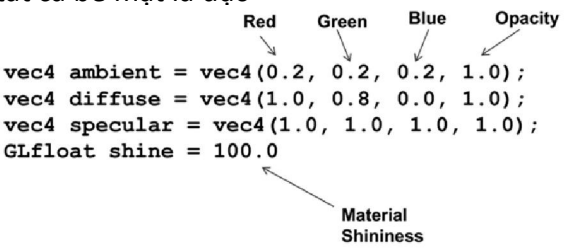
$$m_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y_i + y_{next(i)})$$

Tô bóng củ OpenGL

- Cần:
  - Các vector normal
  - Thuộc tính chất liệu
  - Các nguồn sáng
- Các hàm tô bóng dựa trên trạng thái đã bị bỏ
  - glNormal, glMaterial, glLight

Các thuộc tính chất liệu

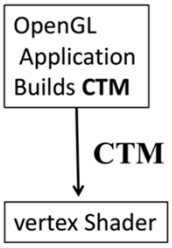
- Cần thiết để đặc tả các thuộc tính chất liệu của các đối tượng trong cảnh.
- Thuộc tính chất liệu cũng có: ambient, diffuse, specular (thuộc tính môi trường, khuếch tán và phản chiếu, chói sáng)
- Thuộc tính chất liệu được đặc tả: RGBA và độ phản xạ.
- Thành phần w – độ trong suốt
- Mặc định: tất cả bề mặt là đặc



Ma trận CTM chuyển vào vertex shader

- Ma trận CTM được kết hợp trong ứng dụng.

  mat4 ctm = ctm * LookAt(vec4 eye, vec4 at, vec4 up)
- CTM chứa cả biến đổi đối tượng + biến đổi camera
- Kết nối với ma trận ModeView trong shader



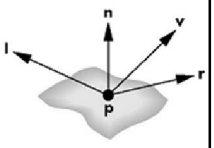

```
in vec4 vPosition;                          CTM passed in
Uniform mat4 ModelView ;

main( )
{
  // Transform vertex  position into eye coordinates
     vec3 pos = (ModelView * vPosition).xyz;
  ……..
}
```

Tính toán các vector

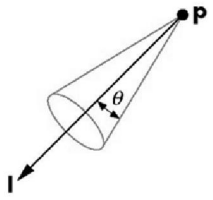
- CTM biến đổi các vị trí đỉnh thành tọa độ mới tương ứng trong hệ tọa độ mắt.
- Chuẩn hóa tất cả các vector
- GLSL có hàm chuẩn hóa: normalize
- Lưu ý: độ dài vector ảnh hưởng đến phép biến đổi tỉ lệ

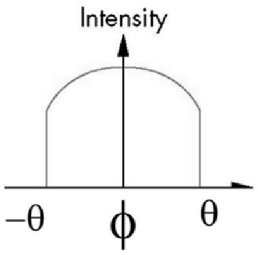
```
// Transform vertex position into eye coordinates
vec3 pos = (ModelView * vPosition).xyz;
vec3 L = normalize( LightPosition.xyz - pos );  // light vector
vec3 E = normalize( -pos );    // view vector
vec3 H = normalize( L + E );    // Halfway vector
```



Spotlight

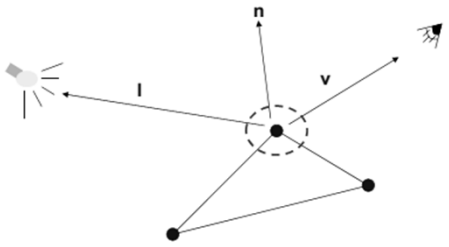
- Được phát triển từ nguồn point light
  - Hướng l
  - Cutoff: θ, không có ánh sáng bên ngoài θ
  - Độ suy giảm: $\cos^{\alpha}\phi$





Chiếu sáng tính cho mỗi đỉnh

- Mô hình chiếu sáng Phong (ambient + diffuse + specular) được tính toán cho mỗi đỉnh để xác định màu sắc đỉnh.
- Tính toán trên mỗi đỉnh như thế nào? Thường xuyên được thực hiện trên vertex shader.



Lập trình Shader chiếu sáng cho mỗi điểm

```
// vertex shader
in vec4 vPosition;
in vec3 vNormal;
out vec4 color;  //vertex shade

// light and material properties
uniform vec4 AmbientProduct, DiffuseProduct, SpecularProduct;
uniform mat4 ModelView;
uniform mat4 Projection;
uniform vec4 LightPosition;
uniform float Shininess;
```

Ambient, diffuse, specular (light * reflectivity) specified by user

$k_a I_a$ $k_d I_d$ $k_s I_s$

exponent of specular term

Lập trình shader chiếu sáng cho mỗi đỉnh (2)

```
void main( )
{
    // Transform vertex position into eye coordinates
    vec3 pos = (ModelView * vPosition).xyz;
    vec3 L = normalize( LightPosition.xyz - pos );
    vec3 E = normalize( -pos );
    vec3 H = normalize( L + E ); // halfway Vector
    // Transform vertex normal into eye coordinates
    vec3 N = normalize( ModelView*vec4(vNormal, 0.0) ).xyz;
```

Lập trình shader chiếu sáng cho mỗi đỉnh (3)

```
    // Compute terms in the illumination equation
    vec4 ambient = AmbientProduct;
    float cos_theta = max( dot(L, N), 0.0 );
    vec4 diffuse = cos_theta * DiffuseProduct;
    float cos_phi = pow( max(dot(N, H), 0.0), Shininess );
    vec4 specular = cos_phi * SpecularProduct;
    if( dot(L, N) < 0.0 ) specular = vec4(0.0, 0.0, 0.0, 1.0);

    gl_Position = Projection * ModelView * vPosition;

    color = ambient + diffuse + specular;
    color.a = 1.0;
}
```

$\leftarrow \mathbf{k_a I_a}$

$\leftarrow \mathbf{k_d I_d}\ \mathbf{l \cdot n}$

$\leftarrow \mathbf{k_s I_s (n \cdot h)^{\beta}}$

$$\mathbf{I = k_d I_d\ l \cdot n \quad + \quad k_s I_s (n \cdot h)^{\beta} \quad + \quad k_a I_a}$$

Lập trình shader chiếu sáng cho mỗi đỉnh (4)

```
// in vertex shader, we declared color as out, set it
…….
color = ambient + diffuse + specular;
color.a = 1.0;
}
// in fragment shader
in vec4 color;
void main()
{
gl_FragColor = color;
}
```