Lighting and Shading In OpenGL

Lighting & Shading Review

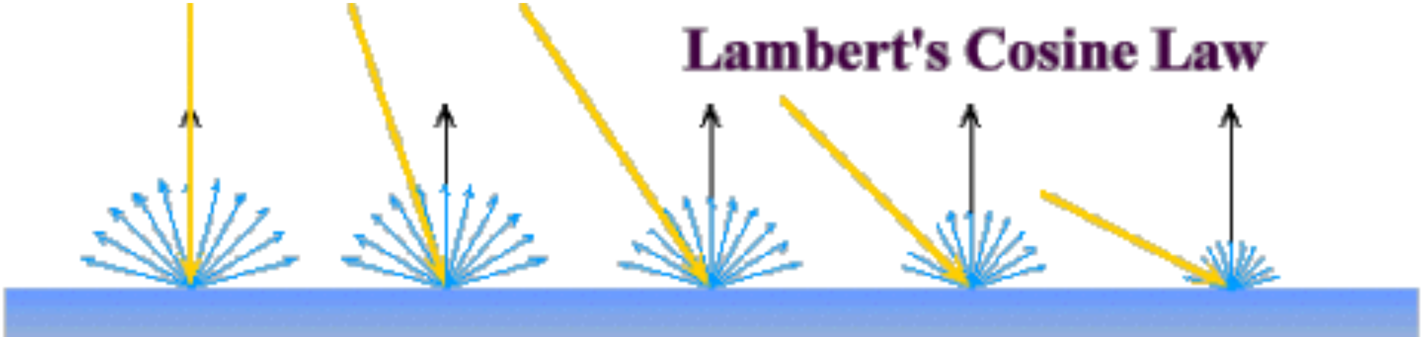
- Lighting Models
  - Ambient
    - Normals don’t matter
  - Lambert/Diffuse
    - Angle between surface normal and light
  - Specular
    - Surface normal, light, and viewpoint

- Gouraud & Phong Shading

- Next Class, OpenGL Lighting & Shading

Lambertian Surface

- Perfectly diffuse reflector
- Light scattered equally in all directions
- Amount of light reflected is proportional to the vertical component of incoming light
  - reflected light: $\sim\cos\theta_i$
  - $\cos\theta_i = l \cdot n$ if vectors normalized
  - There are also three coefficients, k_r, k_b, k_g that show how much of each color component is reflected



Phong Model

- A simple model that can be computed rapidly
- Has three components

  Diffuse

  Specular

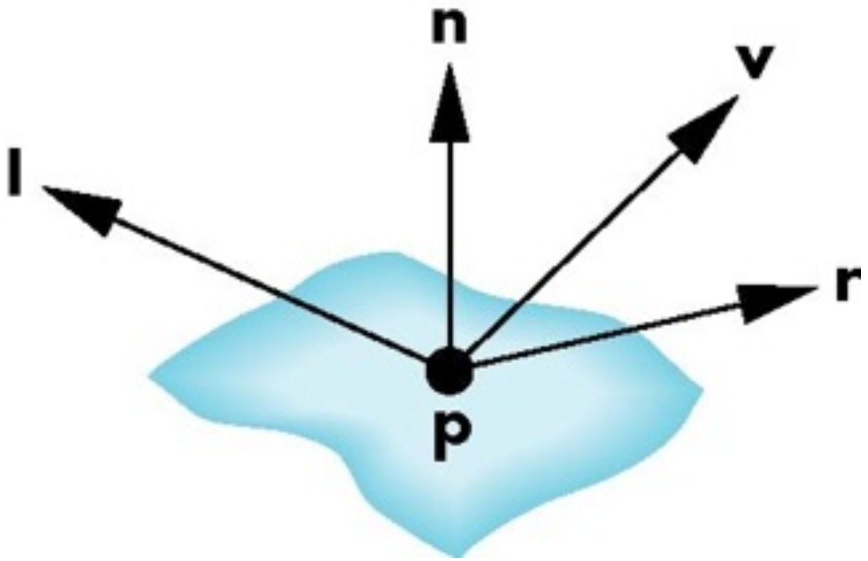
  Ambient
- Uses four vectors

  To light source, **l**

  To viewer, **v**

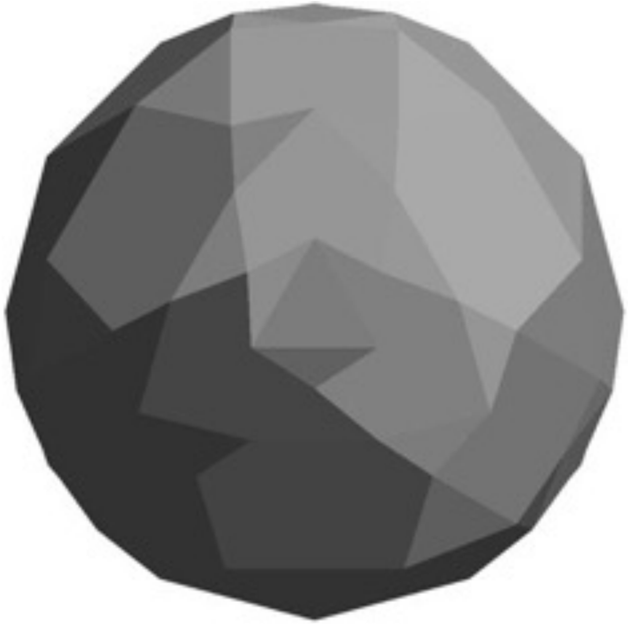
  Normal, **n**

  Perfect reflector, **r**



Polygon Normals

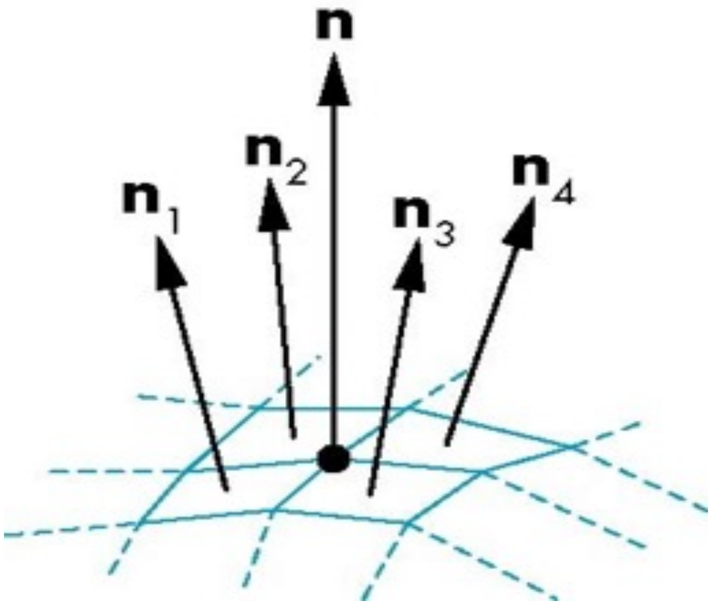
- Polygons have a single normal
  - Shades at the vertices as computed by the Phong model can be almost same
  - Identical for a distant viewer (default) or if there is no specular component
- Consider model of sphere
- Want different normals at each vertex even though this concept is not quite correct mathematically



Mesh Shading

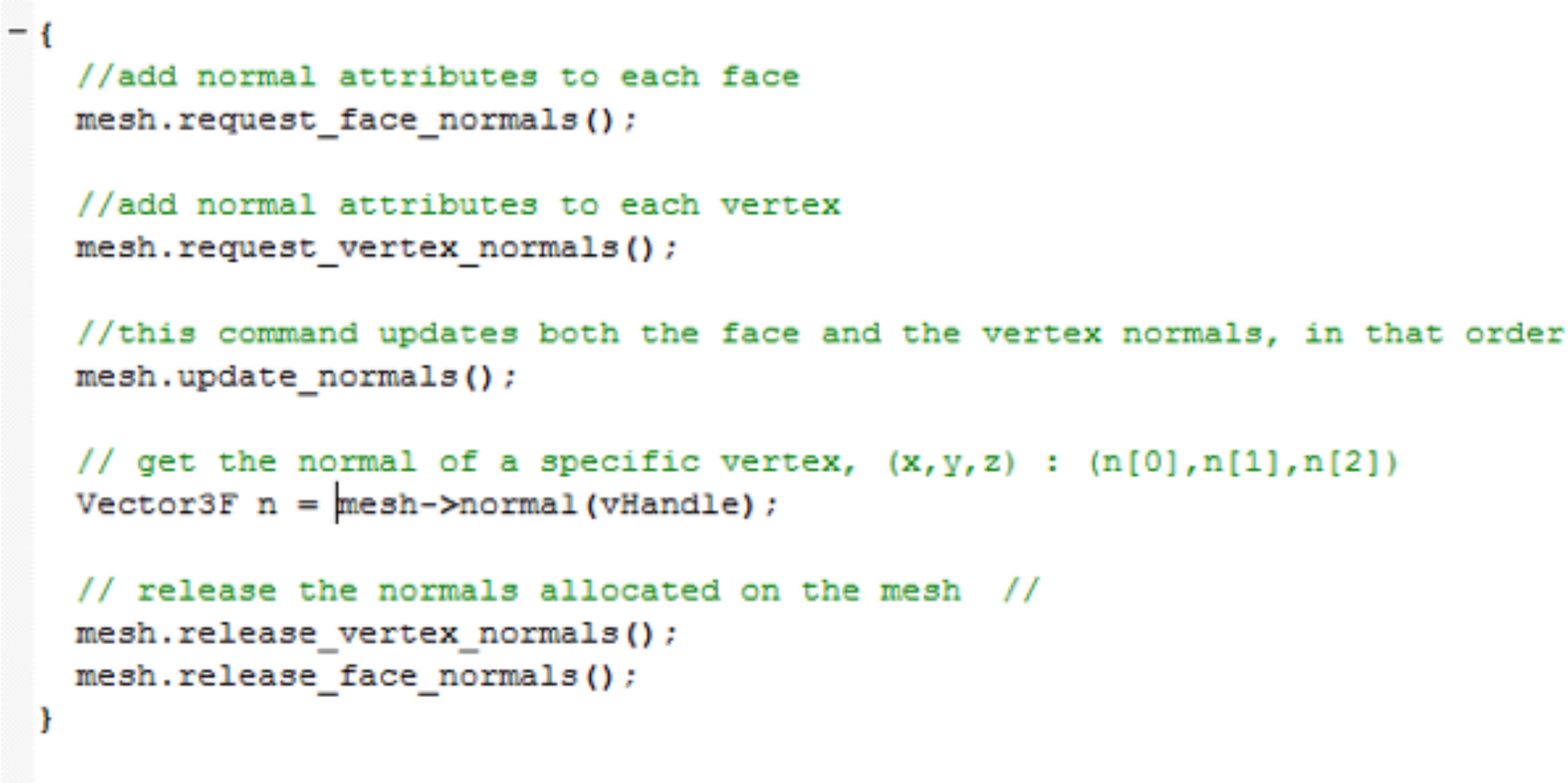
- For polygonal models, Gouraud proposed we use the average of the normals around a mesh vertex

$$n = (n_1 + n_2 + n_3 + n_4) / |n_1 + n_2 + n_3 + n_4|$$



OpenMesh Normals

- OpenMesh can calculate this for you

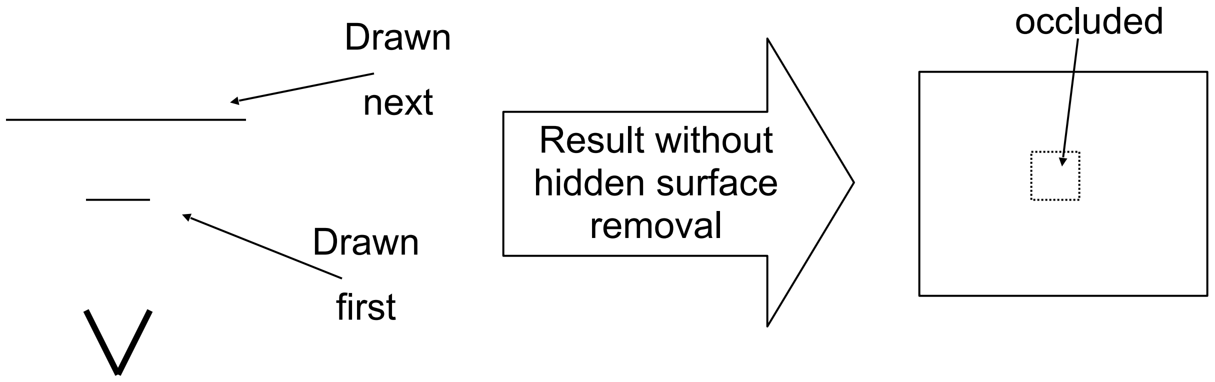


```
{
    //add normal attributes to each face
    mesh.request_face_normals();

    //add normal attributes to each vertex
    mesh.request_vertex_normals();

    //this command updates both the face and the vertex normals, in that order
    mesh.update_normals();

    // get the normal of a specific vertex, (x,y,z) : (n[0],n[1],n[2])
    Vector3F n = mesh->normal(vHandle);

    // release the normals allocated on the mesh  //
    mesh.release_vertex_normals();
    mesh.release_face_normals();
}
```

Hidden Surface Removal

- When drawing objects in order which does not match the order of their appearance (distance from the camera) we get wrong occlusions.

- Note: the order is view dependent, therefore for each viewpoint a different drawing order should be found.



Hidden Surface Removal

- OpenGL solves this problem by holding a depth-map called “Z-Buffer”. This buffer holds the depths (distances on the Z direction) of each pixel drawn on the frame buffer. Then, when a new object is painted, a depth test determines for each pixel if it should be updated or not.
- To turn this mechanism on, the following steps should be taken:

```
glutInitDisplayMode(GLUT_DEPTH | … ) ;
glEnable(GL_DEPTH_TEST) ;
glClear(GL_COLOR_BUFFER_BIT |
GL_DEPTH_BUFFER_BIT) ;
```

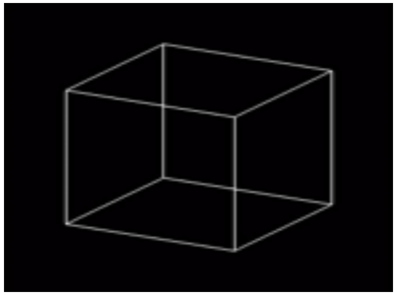
Fogging

Fogging can be used to recreate more natural scenes, by having distant objects merge into the color of the 'background'

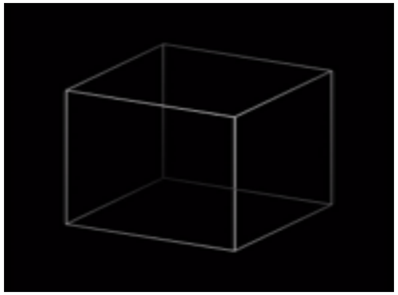
$C = fC_i + f'C_f$ where, $f' = 1 - f$

Ci is the incoming color; Cf is the fog color

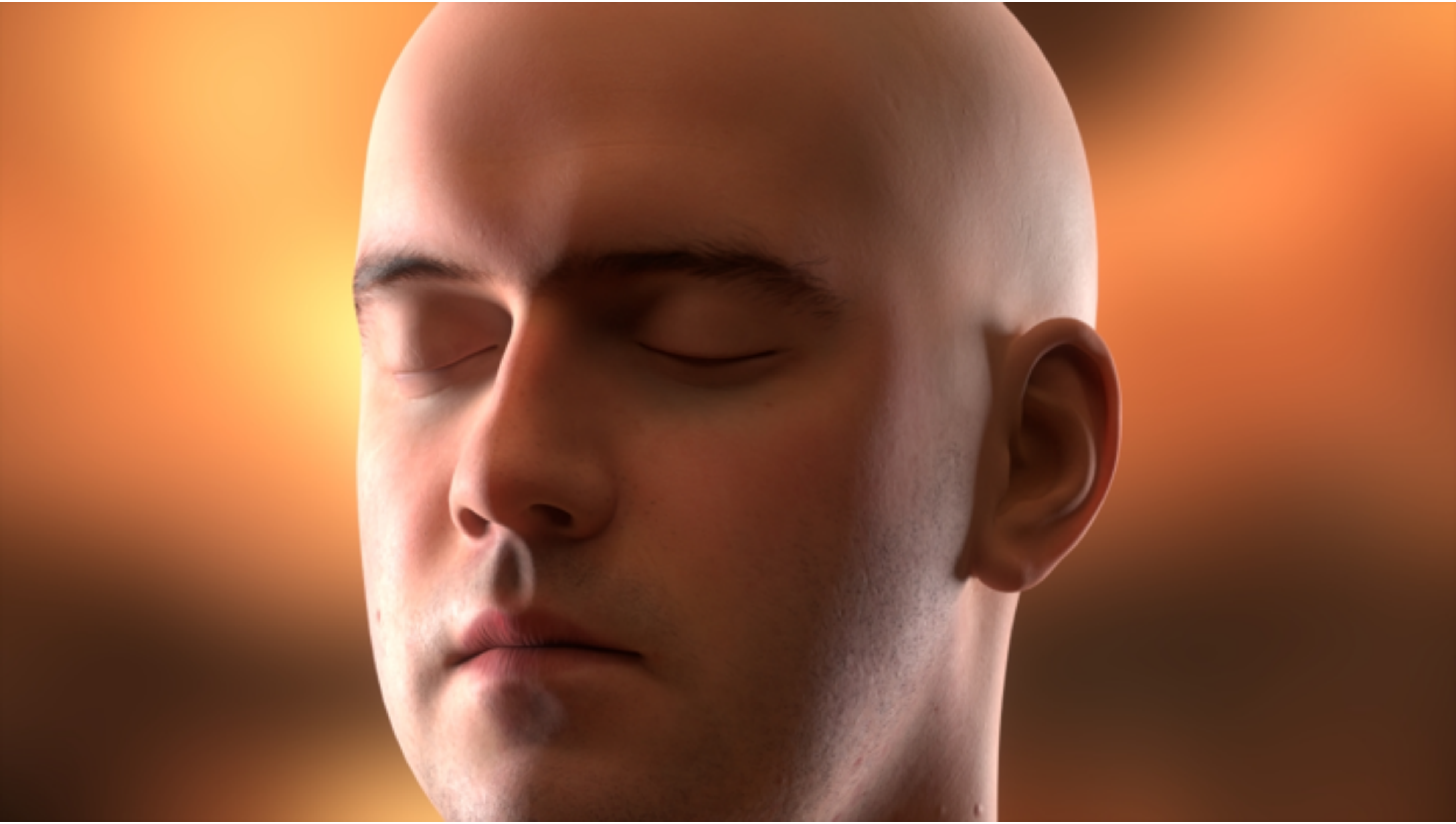
Setting Cf to white would recreate the haziness of looking a large distance through the atmosphere



Setting Cf to black may be used to make near features stand out more



Subsurface Scattering



Subsurface Scattering

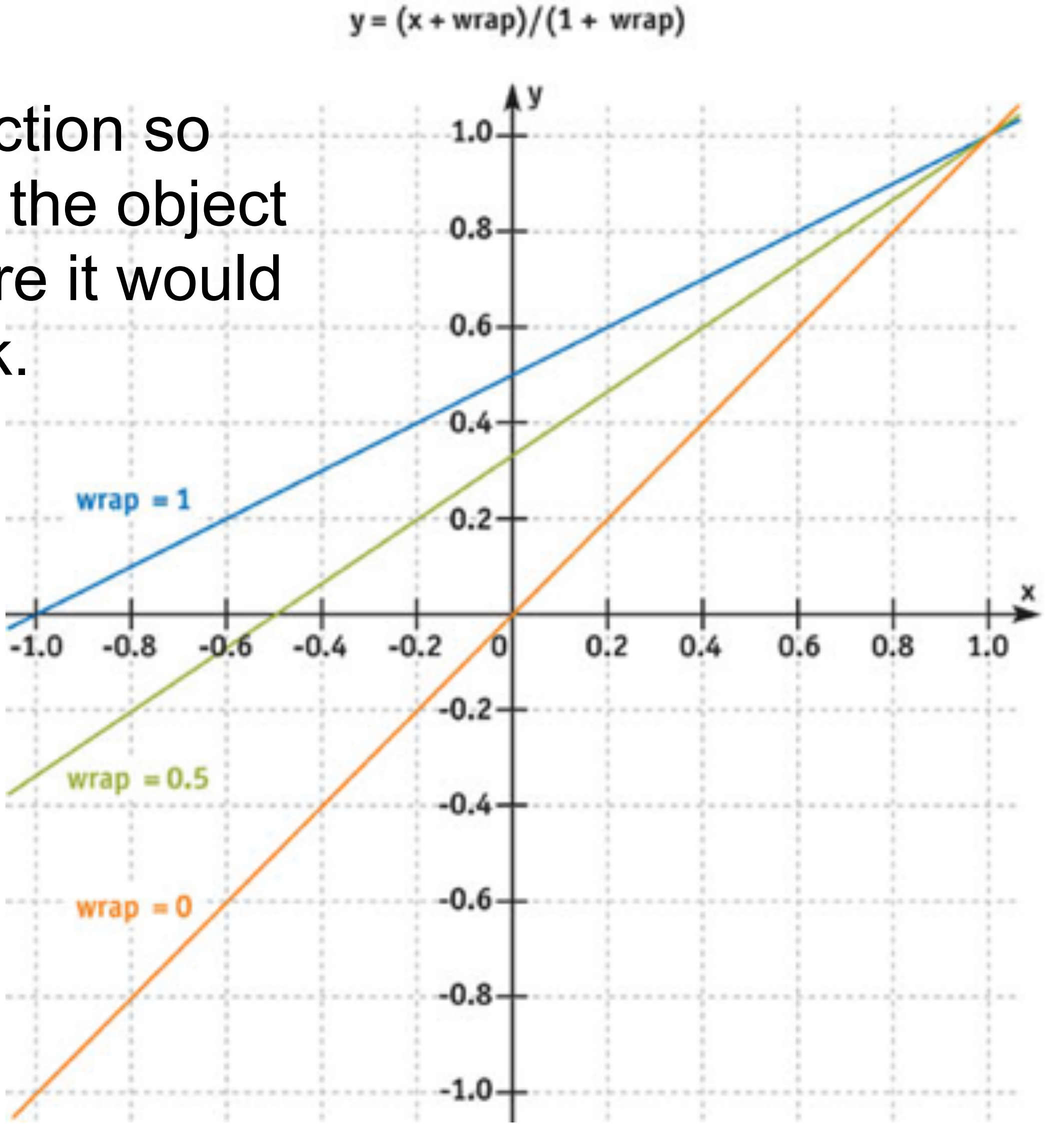
- Light from one area tends to bleed into neighboring areas on the surface
- Small surface details become less visible
- The further the light, the more it attenuated and diffused
- With skin, scattering tends to cause color shift toward red

http://http.developer.nvidia.com/GPUGems/gpugems_ch16.html

Subsurface Scattering

Wrap lighting:

Modify the diffuse function so lighting wraps around the object beyond the point where it would normally become dark.

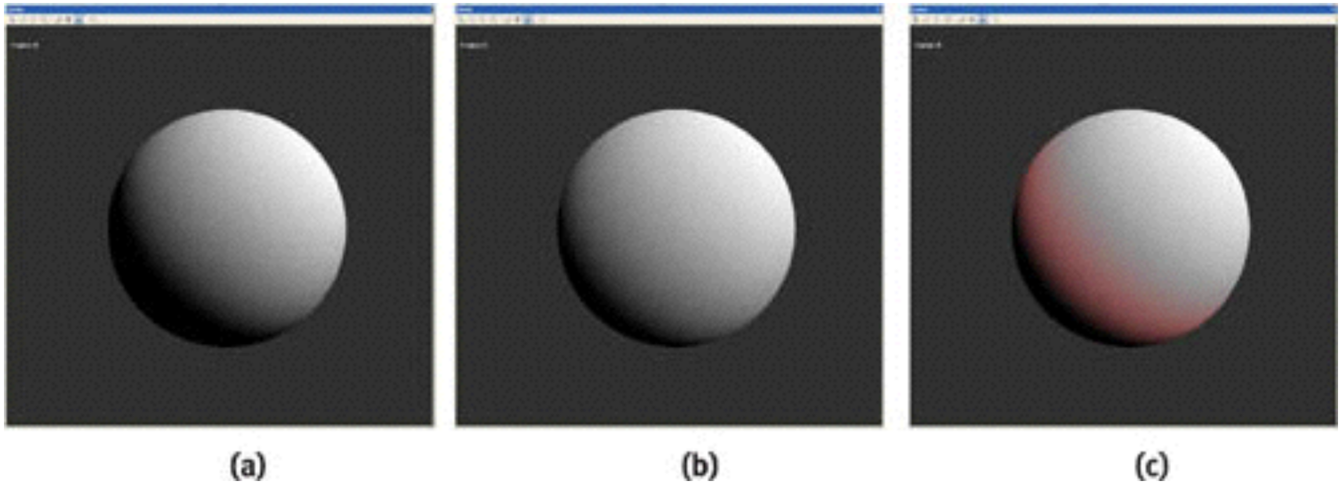


```
float diffuse = max(0, dot(L, N));
float wrap_diffuse = max(0, (dot(L, N) + wrap) / (1 + wrap));
```

Subsurface Scattering

Wrap lighting:

Modify the diffuse function so lighting wraps around the object beyond the point where it would normally become dark.



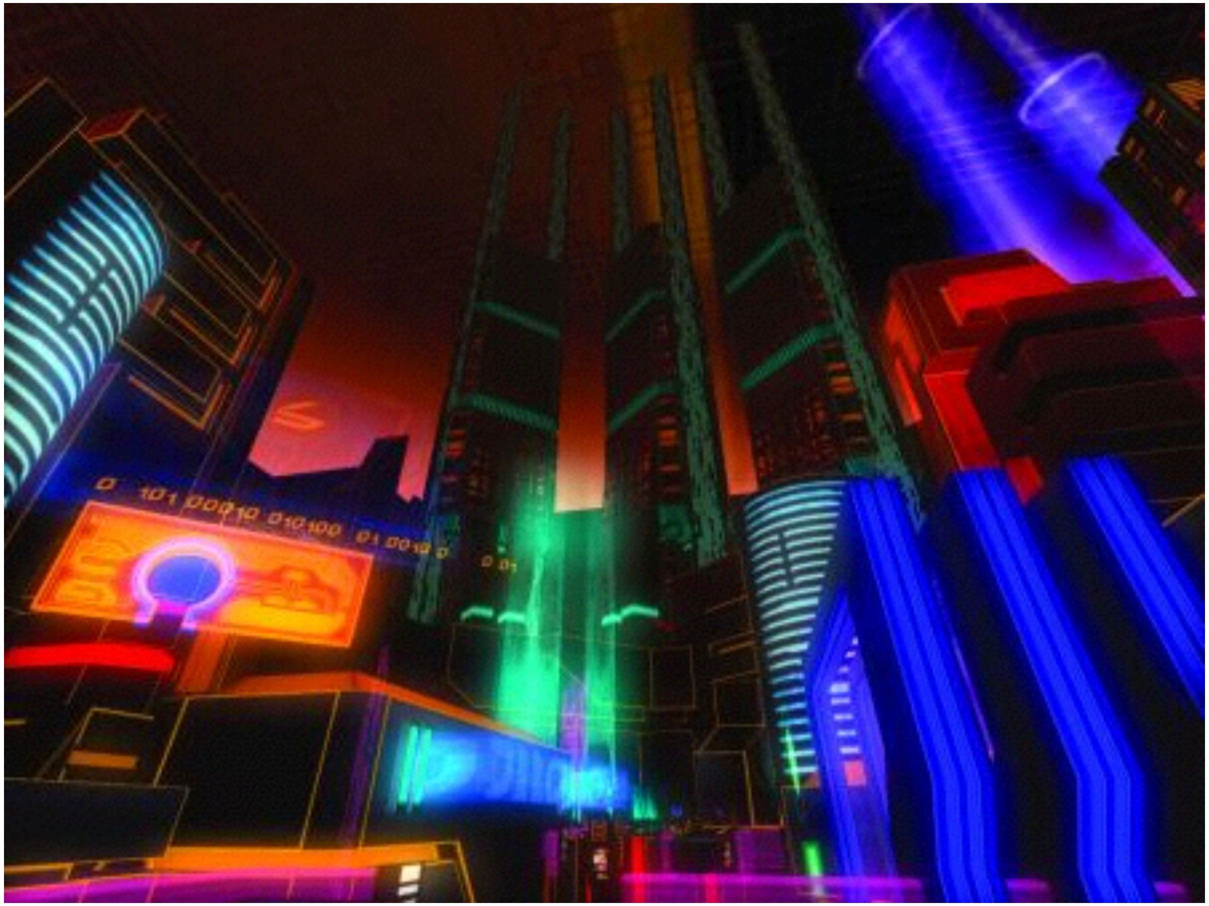
(a)

(b)

(c)

```
float diffuse = max(0, dot(L, N)); // a
float wrap_diffuse = max(0, (dot(L, N) + wrap) / (1 + wrap)); // b
```

Glow

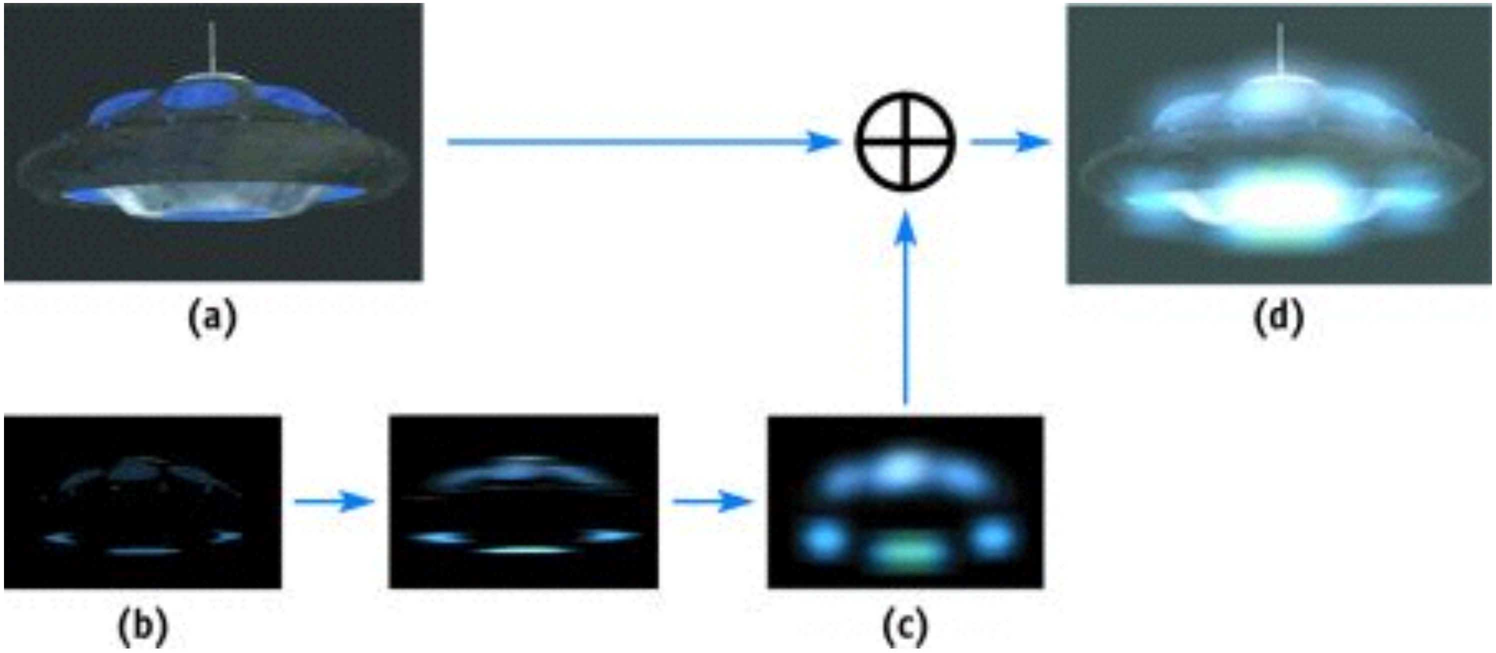


a) The scene is rendered normally.

b) A rendering of glow sources is blurred to create.

c) A glow texture, which is added to the ordinary scene to produce.

d) The final glow effect.



http://devmaster.net/posts/3100/shader-effects-glow-and-bloom

http://www.gamasutra.com/view/feature/2107/realtime_glow.php