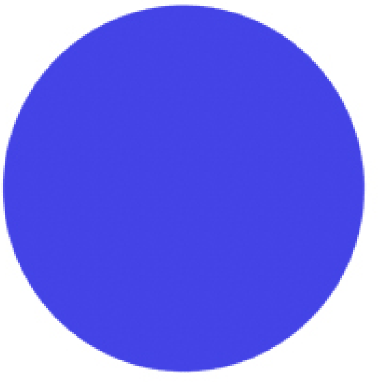
3D Graphics Programming Tools

Lighting

Today's agenda

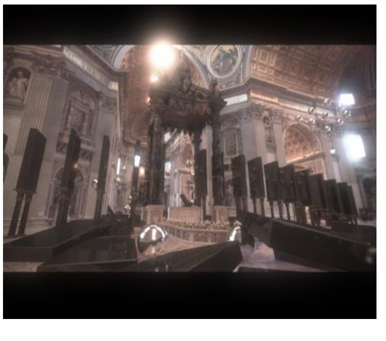
- Why do we need to calculate lighting?
- Definitions
- Ambient and directional light sources
- Diffuse and specular reflection
- OpenGL and lighting

Lighting versus “Colouring”



Solving the lighting problem

- Where are we?
  - We somewhat understand the perception of light (colour)
  - We know how to represent and generate colour using computers
- We now need to understand the interplay of light and objects



Rendering with natural light

Light stage

Lighting

- Later, we will learn how to rasterise
  - i.e., given a 3-D triangle and a 3-D viewpoint, we know which pixels represent the triangle
- But … what colour should those pixels be?

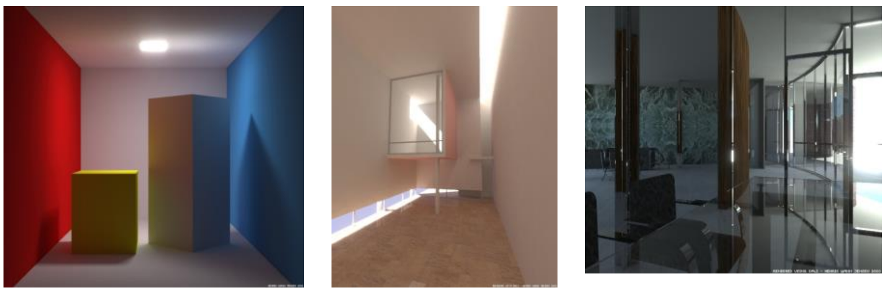
- To create a realistic image → need to simulate the lighting of the surfaces in the scene
  - Fundamentally → simulation of *physics* and *optics*
  - In reality → we use a lot of approximations (perceptually based hacks) to do this simulation fast enough

Today's agenda

- Why do we need to calculate lighting?
- Definitions
- Ambient and directional light sources
- Diffuse and specular reflection
- OpenGL and lighting

Definitions

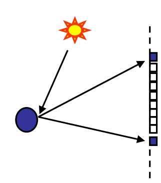
- Illumination
  - the transport of energy from light sources to surfaces & points
    - Note: includes *direct* and *indirect illumination*



[Images by Henrik Wann Jensen]

Definitions

- Lighting
  - the process of computing the luminous intensity (i.e. outgoing light) at a particular 3-D point
- Shading
  - the process of assigning colours to pixels



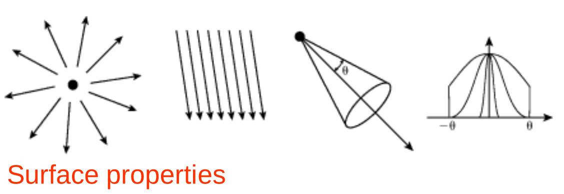
Definitions

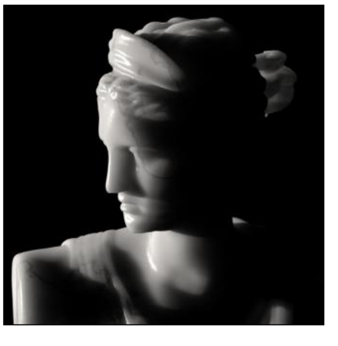
- Illumination models → two categories:
  - Empirical
    - simple formulations that approximate observed phenomenon
  - Physically based
    - models based on the actual physics of light interacting with matter

- Interactive graphics
  - for simplicity → mostly use empirical models
  - increasingly → realistic graphics are using physically based models

Two components of lighting

- Light sources (or emitters)
  - spectrum of emittance (*colour of the light*)
  - geometric attributes (*position*, *direction*, *shape*)
  - directional attenuation

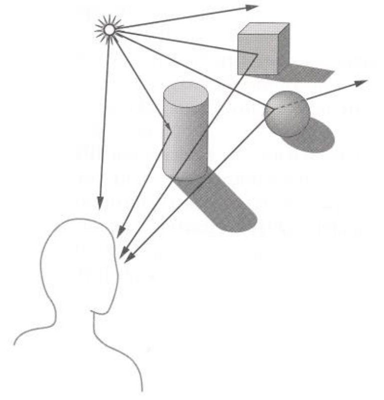




- Surface properties
  - reflectance spectrum (*colour of the surface*)
  - subsurface reflectance
  - geometric attributes (*position*, *orientation*, *micro-structure*)

Goal

- Must derive computer models for ...
  - Emission at light sources
  - Scattering at surfaces
  - Reception at the camera



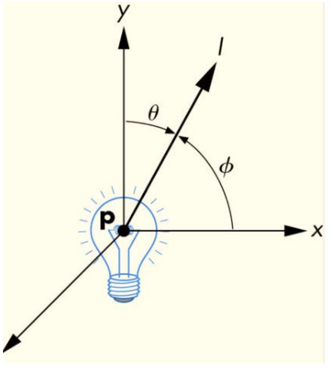
- Desirable features …
  - Concise
  - Efficient to compute
  - "Accurate"

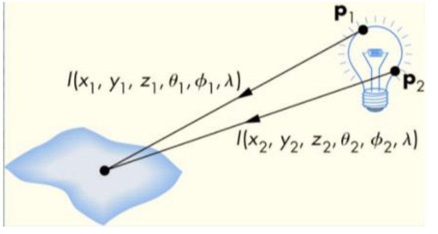
Today's agenda

- Why do we need to calculate lighting?
- Definitions
- Ambient and directional light sources
- Diffuse and specular reflection
- OpenGL and lighting

Modelling light sources

- I $(x,y,z,\theta,\phi,\lambda)$ ...
  - describes the intensity of energy (I)
  - leaving a light source from location(x,y,z)
  - with direction (θ,ϕ)
  - at wavelength λ



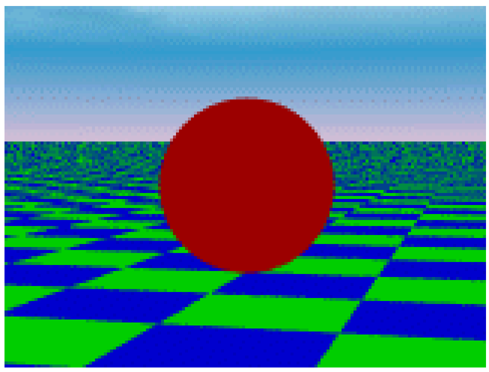


Ambient light sources

- Simulate Indirect illumination from emitters, bouncing off intermediate surfaces
  - Objects not directly lit are typically still visible
    - e.g., the ceiling of a room, undersides of desks
  - Too expensive to calculate (in real time), so we use a hack called an ambient light source
    - No spatial or directional characteristics
    - Illuminates all surfaces equally
    - Amount reflected depends on surface properties

Ambient light sources

- A scene lit only with an ambient light source



Light position not important

Viewer position not important

Surface angle not important

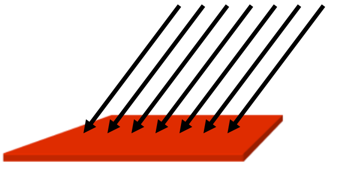
Ambient term

- Represents reflection of all indirect illumination

Note: this is a hack (avoids complexity of global illumination)!

Distant light sources

- For a distant light source we make simplifying assumptions
  - direction is constant for all surfaces in the scene
  - all rays of light from the source are parallel
    - As if the source were infinitely far away from the surfaces in the scene
    - A good approximation to sunlight
- The direction from a surface to the light source is important in lighting the surface

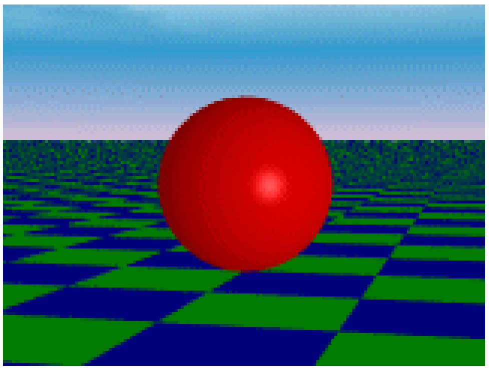


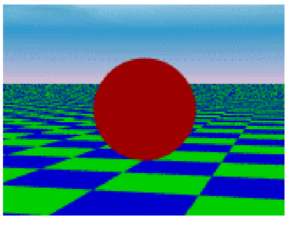
EBU5405

19

Distant light sources

- The same scene lit with a distant and an ambient light source





EBU5405

20

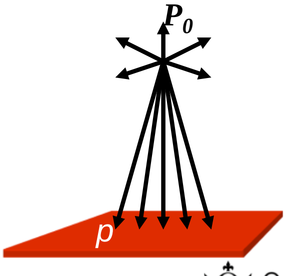
Point light sources

- Point light source
  - emits light equally in all directions from a single point
  - the direction to the light from a point on a surface differs for different points → need to calculate a normalised vector to the light source for every point we light

$$\bar{d} = \frac{\bar{p} - \bar{l}}{\left\| \bar{p} - \bar{l} \right\|}$$

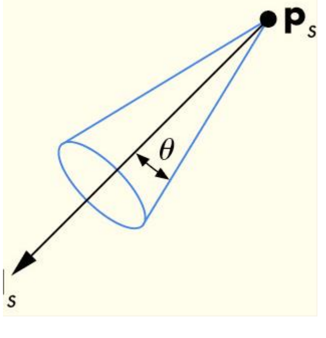
  - The intensity of illumination received from a source located at P0 at a point P is proportional to the inverse square of the distance from the source

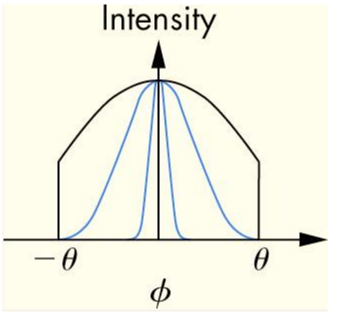
$$L(P, P_0) = \frac{1}{|P - P_0|^2} L(P_0)$$



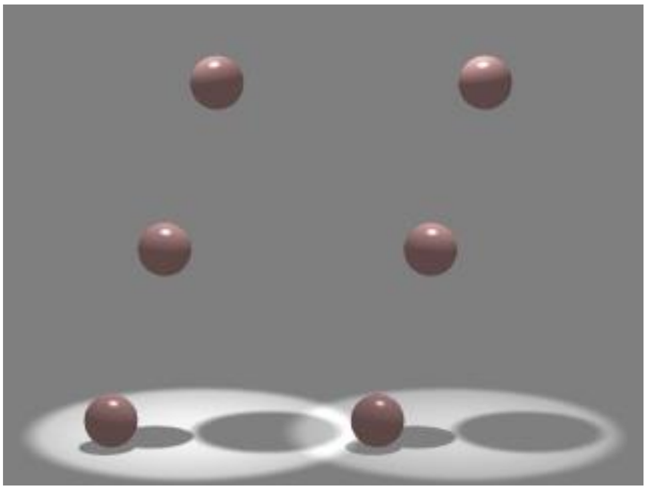
Spotlights

- Spotlights
  - Characterised by a narrow range of angles through which light is emitted.
  - Realistic spotlights are characterised by the distribution of light within the cone (usually with most of the light concentrated in the center of the cone).



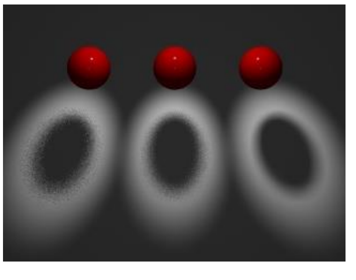


Spotlights



Other light sources

- Area light sources
  - 2-D emissive surface (usually a disc or polygon)
  - example: fluorescent light panels
  - capable of generating soft shadows

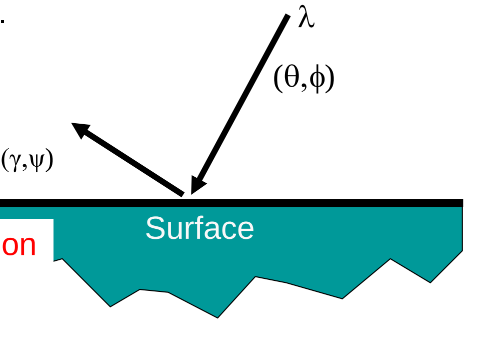


Today’s agenda

- Definitions
- Ambient and directional light sources
- Diffuse and specular reflection
- OpenGL and lighting

Modelling surface reflectance

- $R_s(\theta,\phi,\gamma,\psi,\lambda)$ ... describes the amount of incident energy:
  - arriving from direction (θ,ϕ), ...
  - leaving in direction (γ,ψ), …
  - with wavelength λ

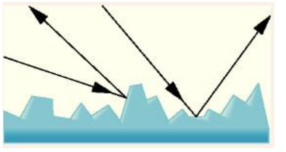


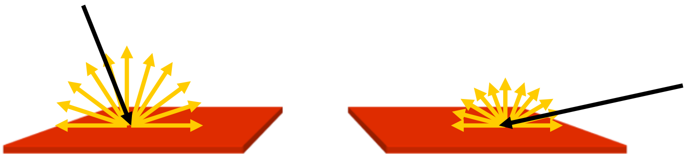
Bi-directional reflection distribution (BRDF)

- Ideally
  - measure radiant energy for “all” combinations of incident angles
    - too much storage
    - difficult in practice

The physics of reflection

- Ideal diffuse reflection
  - is a very rough surface at the microscopic level
  - an incoming ray of light is equally likely to be reflected in any direction over the hemisphere:





  - The amount of light reflected depends on angle of incident light
    - Lambert's cosine law

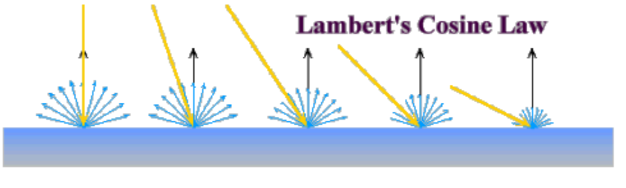
EBU5405

Queen Mary University of London

27

Lambert's cosine law

- Ideal diffuse surfaces reflect according to Lambert's cosine law
  - The energy reflected by a small portion of a surface from a light source in a given direction is proportional to the cosine of the angle between that direction and the surface normal
  - These are often called Lambertian surfaces
  - Note that the reflected intensity is independent of the viewing direction, but does depend on the surface orientation with regard to the light source



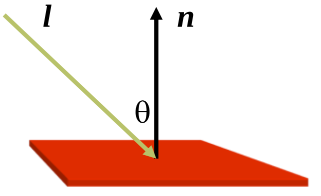
EBU5405

Queen Mary University of London

28

Computing diffuse reflection

- The angle between the surface normal and the incoming light is the *angle of incidence*:



- $I_{diffuse} = k_d\, I_{light}\, \mathbf{cos}\ \theta$

In practice we use vector arithmetic:

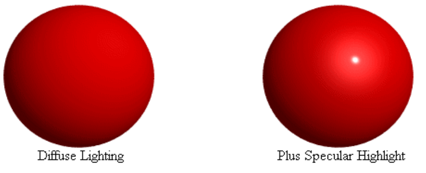
- $I_{diffuse} = k_d\, I_{light}\ (\boldsymbol{n} \bullet \boldsymbol{l})$

Diffuse lighting examples

- A Lambertian sphere seen at several different lighting angles:

Specular reflection

- Shiny surfaces exhibit specular reflection
  - Polished metal
  - Glossy car finish

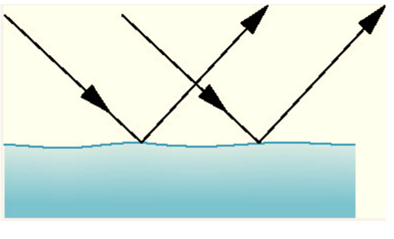


- A light shining on a specular surface causes a bright spot (*specular highlight*)
- Where these highlights appear is a function of the viewer's position
  → specular reflectance is view dependent

The physics of reflection

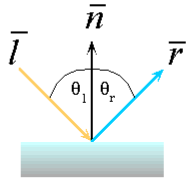
- Specular reflecting surface
  - is very smooth at the microscopic level
  - rays of light → likely to bounce off the micro-geometry in a mirror-like fashion
  - The smoother the surface, the closer it becomes to a perfect mirror

  - Reflection is strongest near mirror angle



The optics of reflection

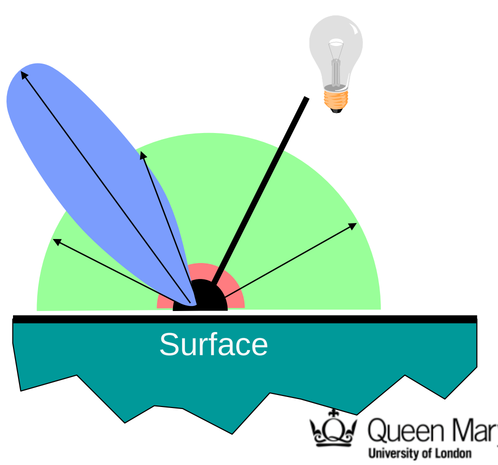
– The incoming ray and reflected ray lie in a <u>plane</u> with the surface normal
– The angle that the reflected ray forms with the surface normal equals the angle formed by the incoming ray and the surface normal:



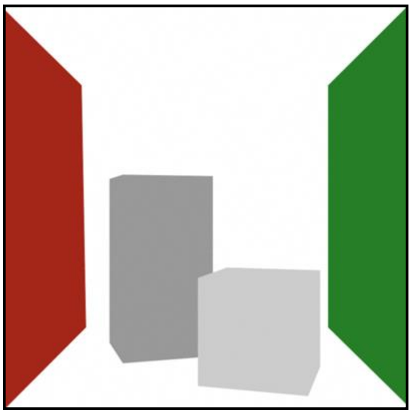
$$\theta_{\text{(l)ight}} = \theta_{\text{(r)eflection}}$$

Combining everything

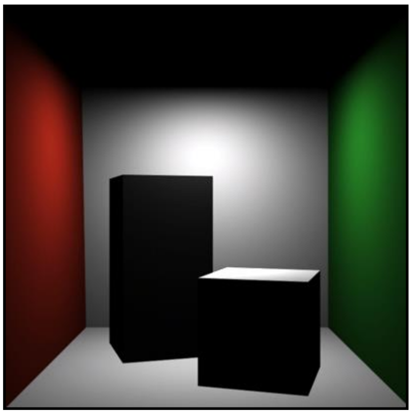
- Simple analytic model:
  – diffuse reflection +
  – specular reflection +
  – emission +
  – “ambient”



Lighting example: diffuse reflection

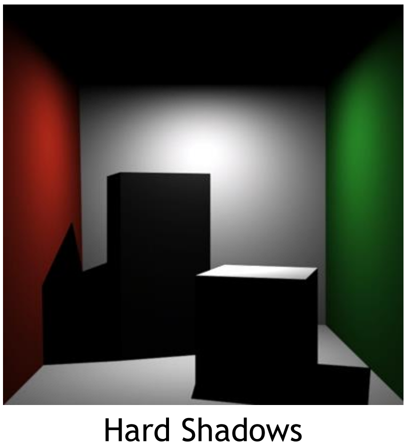


Surface Color



Diffuse Shading

Lighting example: soft shadows



Hard Shadows
Point Light Source

Soft Shadows
Area Light Source

Direct illumination: summary

- Model for
  - determining the brightness (radiance) of a ray rooted at a point on a surface and oriented towards the camera
    - Ambient term
    - Diffuse term
    - Specular term

- Influencing factors
  - Light position
  - Sample point position
  - Camera position
  - Surface angle with respect to light vector
  - Surface angle with respect to camera vector

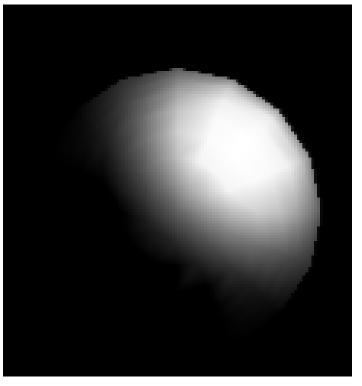
Direct illumination: questions

- Camera moves from one position to another
  - Angle between light and surface unchanged
  - Angle between camera and surface changes
- A tracking camera follows object as it moves in scene
  - Angle between light and surface changes
  - Angle between camera and surface unchanged
- An object moves from one position to another
  - Both angles change

Some questions …

- Consider the image of a sphere shown below.

Discuss the lighting conditions under which the sphere has been rendered.

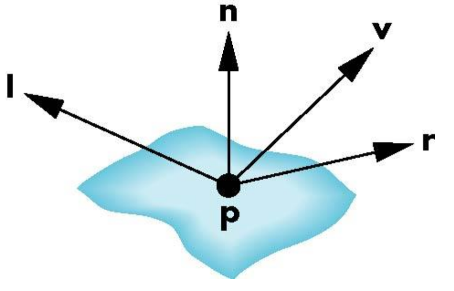


Today's agenda

- Definitions
- Ambient and directional light sources
- Diffuse and specular reflection
- OpenGL and lighting

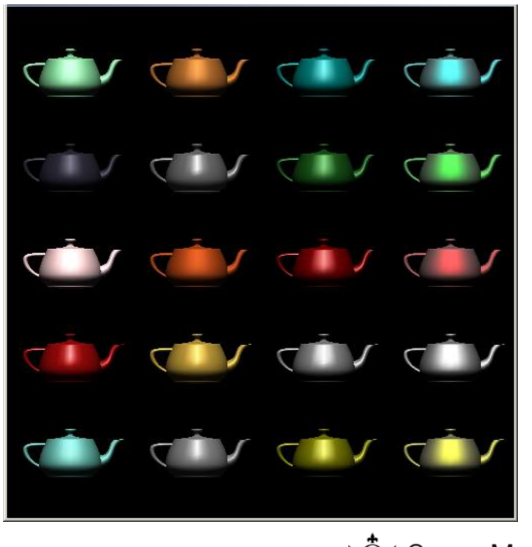
Phong Model

- A simple model that can be computed rapidly
- Has three components
  - –Diffuse
  - –Specular
  - –Ambient
- Uses four vectors
  - –To source (l)
  - –To viewer (v)
  - –Normal (n)
  - –Perfect reflector (r)



Example

Only differences in these teapots are the parameters in the modified Phong model



Steps in OpenGL shading

1. Enable shading and select model
2. Specify normals
3. Specify lights
4. Specify material properties

Enabling Shading

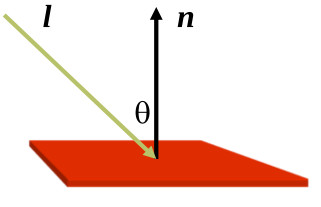
- Shading calculations are enabled by
  - `glEnable(GL_LIGHTING)`
  - Once lighting is enabled, glColor() is ignored!!
- Must enable each light source individually
  - `glEnable(GL_LIGHTi)` i=0,1…..
- Can choose light model parameters to control the shading calculations
  - `glLightModeli(parameter, GL_TRUE)`
    - `GL_LIGHT_MODEL_LOCAL_VIEWER` do not use simplifying distant viewer assumption in calculation
    - `GL_LIGHT_MODEL_TWO_SIDED` shades both sides of polygons independently
  - `glLightModelf(GL_LIGHT_MODEL_AMBIENT, global_ambient);`
    - To create a small amount of ambient light even when all the sources are turned off.
    - GLfloat global_ambient[] = {1.0, 1.0, 1.0, 1.0};

Normals

- In OpenGL the normal vector is part of the state
- Set by **`glNormal*()`**
  - **`glNormal3f(x, y, z);`**
  - **`glNormal3fv(p);`**
- Usually we want to set the normal to have unit length so cosine calculations are correct
  - Length can be affected by transformations
  - Note that scaling does not preserve length
  - **`glEnable(GL_NORMALIZE)`** allows for autonormalization at a performance penalty

Computing diffuse reflection

- The angle between the surface normal and the incoming light is the *angle of incidence:*



- $I_{diffuse} = k_d \, I_{light} \, \mathbf{cos}\, \theta$

In practice we use vector arithmetic:

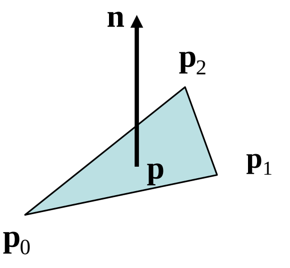
- $I_{diffuse} = k_d \, I_{light} \, (\boldsymbol{n} \bullet \boldsymbol{l})$

Normal for a Triangle

plane $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

$\mathbf{n} = (\mathbf{p}_2 - \mathbf{p}_0) \times (\mathbf{p}_1 - \mathbf{p}_0)$

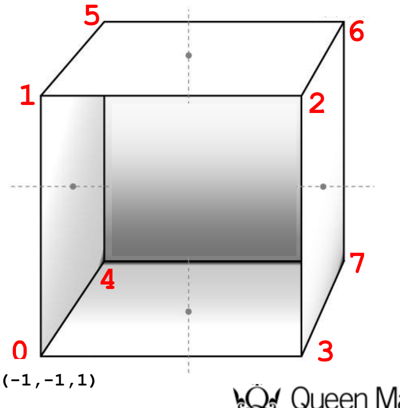
normalize $\mathbf{n} \leftarrow \mathbf{n} / |\mathbf{n}|$

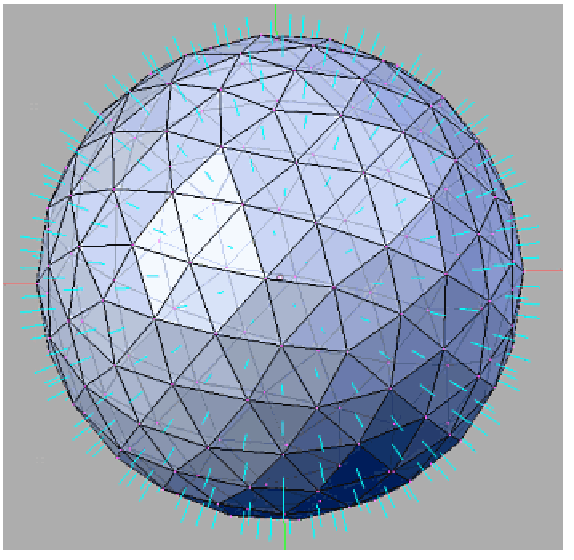


Note that right-hand rule determines outward face

Example (rotating cube)

```
GLfloat GlobalVertices[][3] = {{-1.0,-1.0,1.0}, {-1.0,1.0,1.0}, {1.0,1.0,1.0},
    {1.0,-1.0,1.0}, {-1.0,-1.0,-1.0}, {-1.0,1.0,-1.0}, {1.0,1.0,-1.0},  {1.0,-1.0,-1.0}};

GLfloat normals[][3] = {{0.0, 0.0, 1.0}, {1.0, 0.0, 0.0},
  {0.0, -1.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, -1.0}, {-1.0, 0.0, 0.0}};

void colorcube()
{
  glNormal3fv(normals[0]);
  a3dpolygon(CubeVertices, 0,3,2,1);
  glNormal3fv(normals[1]);
  a3dpolygon(CubeVertices, 2,3,7,6);
  glNormal3fv(normals[2]);
  a3dpolygon(CubeVertices, 3,0,4,7);
  glNormal3fv(normals[3]);
  a3dpolygon(CubeVertices, 1,2,6,5);
  glNormal3fv(normals[4]);
  a3dpolygon(CubeVertices, 4,5,6,7);
  glNormal3fv(normals[5]);
  a3dpolygon(CubeVertices, 5,4,0,1);
}
```





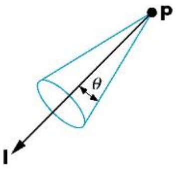
Defining a Point Light Source

- For each light source, we can set an RGBA for the diffuse, specular, and ambient components
- The position is given in homogeneous coordinates
  - If w =1.0, we are specifying a finite location
  - If w =0.0, we are specifying a parallel source with the given direction

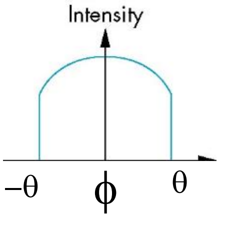
```
GLfloat diffuse0[]={1.0, 0.0, 0.0, 1.0};
GLfloat ambient0[]={1.0, 0.0, 0.0, 1.0};
GLfloat specular0[]={1.0, 0.0, 0.0, 1.0};
Glfloat light0_pos[]={1.0, 2.0, 3,0, 1.0};

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightv(GL_LIGHT0, GL_POSITION, light0_pos);
glLightv(GL_LIGHT0, GL_AMBIENT, ambient0);
glLightv(GL_LIGHT0, GL_DIFFUSE, diffuse0);
glLightv(GL_LIGHT0, GL_SPECULAR, specular0);
```

Spotlights

- Use **glLightv** to set
  - Direction **GL_SPOT_DIRECTION**
  - Angle **GL_SPOT_CUTOFF**
  - Attenuation **GL_SPOT_EXPONENT**
    - Proportional to $\cos^{\alpha}\phi$



- glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, direction);
- glLightfv(GL_LIGHT0, GL_SPOT_CUTOFF, cutoff);

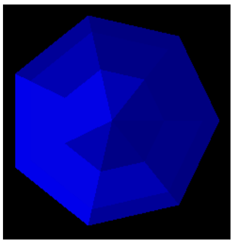


Global Ambient Light

- Ambient light depends on color of light sources
  - A red light in a white room will cause a red ambient term that disappears when the light is turned off
- OpenGL also allows a global ambient term that is often helpful for testing
  - **glLightModelfv(GL_LIGHT_MODEL_AMBIENT, global_ambient)**

Light Sources and Transformations

- Light sources are geometric objects whose positions or directions are affected by the model-view matrix
- Depending on where we place the position (direction) setting function in the program, we can
  - Move the light source(s) with the object(s)
  - Fix the object(s) and move the light source(s)
  - Fix the light source(s) and move the object(s)
  - Move the light source(s) and object(s) independently



Light Sources and Transformations

Example:

```
Void init()
{
Glfloat light_pos[] = {1.0, 2.0, 3.0, 1.0};

/* rest of init */

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
}
```

Material Properties

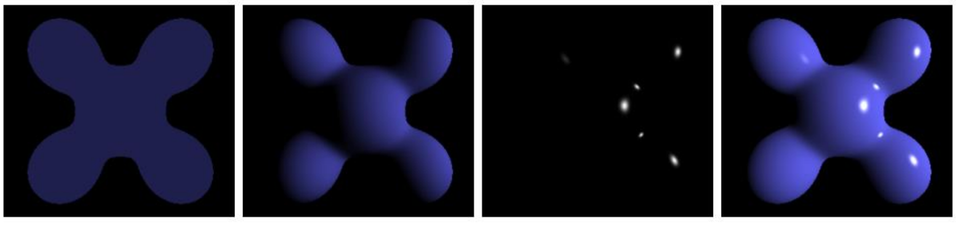
- Material properties are also part of the OpenGL state
- Set by **glMaterialv()**

```
GLfloat ambient[] = {0.2, 0.2, 0.2, 1.0};
GLfloat diffuse[] = {0.8, 0.8, 0.8, 1.0};
GLfloat specular[] = {0.0, 0.0, 0.0, 1.0};
GLfloat shine[] = {100.0};
glMaterialf(GL_FRONT, GL_AMBIENT, ambient);
glMaterialf(GL_FRONT, GL_DIFFUSE, diffuse);
glMaterialf(GL_FRONT, GL_SPECULAR, specular);
glMaterialf(GL_FRONT, GL_SHININESS, shine);
```

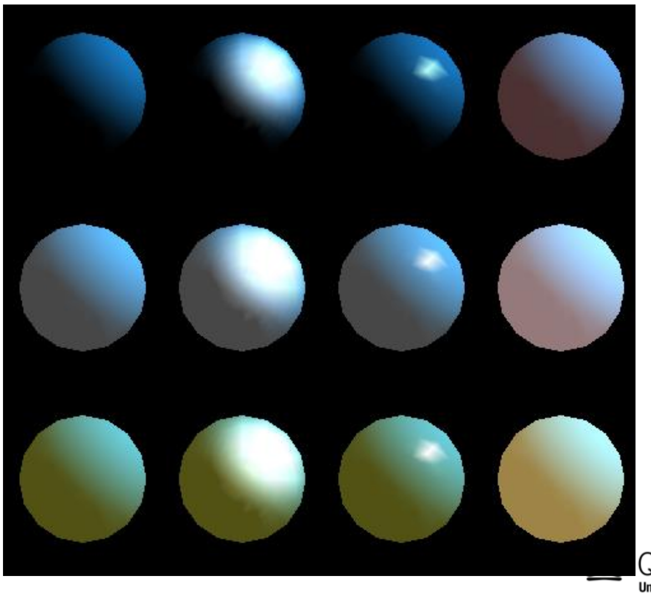

Emissive Term

- We can simulate a light source in OpenGL by giving a material an emissive component
- This component is unaffected by any sources or transformations

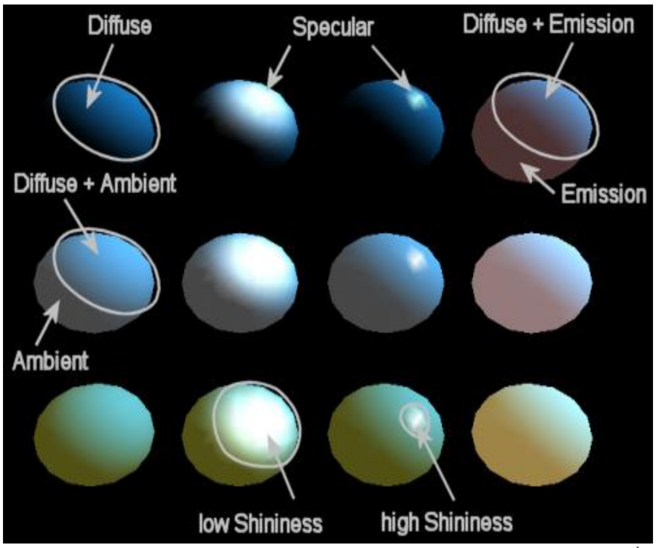
```
GLfloat emission[] = {0.0, 0.3, 0.3, 1.0};
glMaterialf(GL_FRONT, GL_EMISSION, emission);
```



Spheres example



Material Properties



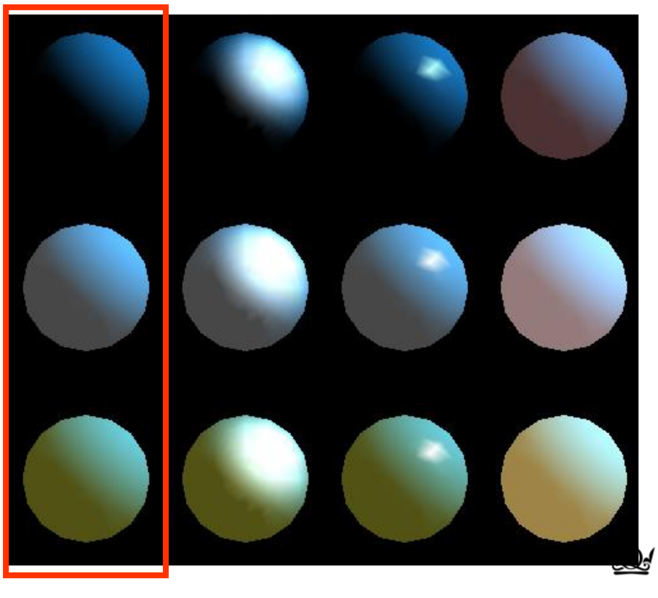
Light source

```
float ambient[] = {0.0, 0.0, 0.0, 1.0};
float diffuse[] = {1.0, 1.0, 1.0, 1.0};
float specular[] = {1.0, 1.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, specular);

 //light model properties
float model_ambient[] = {0.4, 0.4, 0.4, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, model_ambient);
```

Material properties

```
float no_mat[] = {0.0, 0.0, 0.0, 1.0};
float mat_ambient[] = {0.7, 0.7, 0.7, 1.0};
float mat_ambient_color[] = {0.8, 0.8, 0.2, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
float no_shininess = 0.0;
float low_shininess = 5.0;
float high_shininess = 100.0;
float mat_emission[] = {0.3, 0.2, 0.2, 1.0};
```

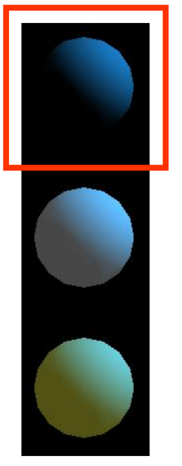

Material properties



Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
Light: float diffuse[] = {1.0, 1.0, 1.0, 1.0};



Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
Light: float diffuse[] = {1.0, 1.0, 1.0, 1.0};

float mat_ambient[] = {0.7, 0.7, 0.7, 1.0};
float model_ambient[] = {0.4, 0.4, 0.4, 1.0};

Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);

glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);

glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);

glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

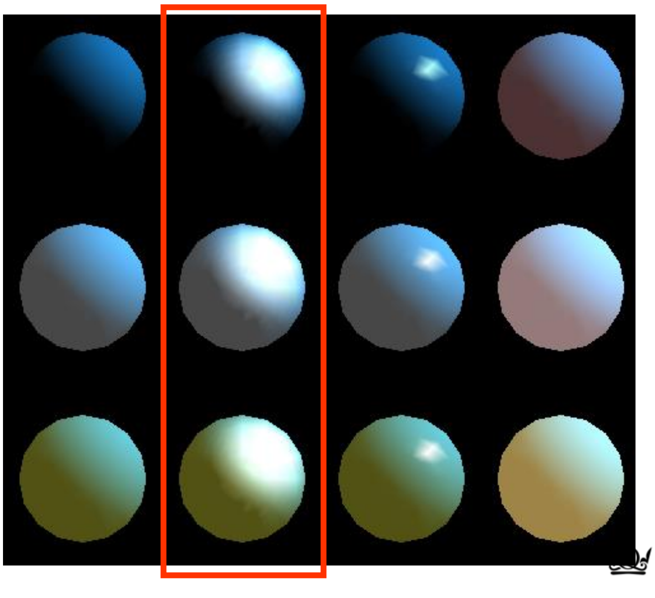
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};

Light: float diffuse[] = {1.0, 1.0, 1.0, 1.0};

float mat_ambient_color[] = {0.8, 0.8, 0.2, 1.0};

float model_ambient[] = {0.4, 0.4, 0.4, 1.0};

Material properties



Material properties

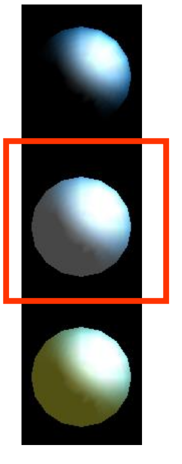
glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
Light: float diffuse[] = {1.0, 1.0, 1.0, 1.0};

float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
float specular[] = {1.0, 1.0, 1.0, 1.0};

Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);



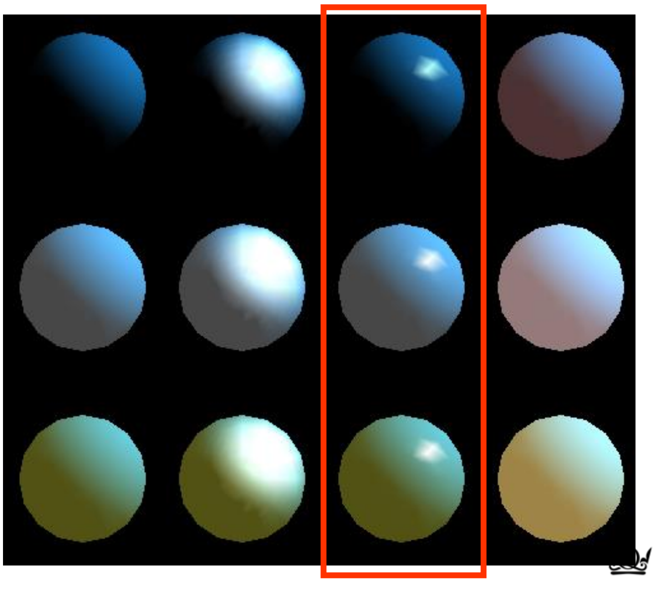
float mat_ambient[] = {0.7, 0.7, 0.7, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};

Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_ambient_color[] = {0.8, 0.8, 0.2, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};

Material properties

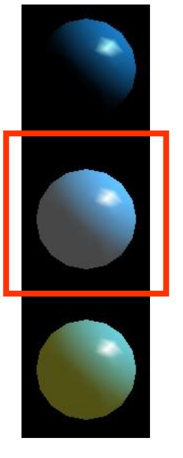


Material properties

```
glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, high_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
```

Material properties

```
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, high_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_ambient[] = {0.7, 0.7, 0.7, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};
```

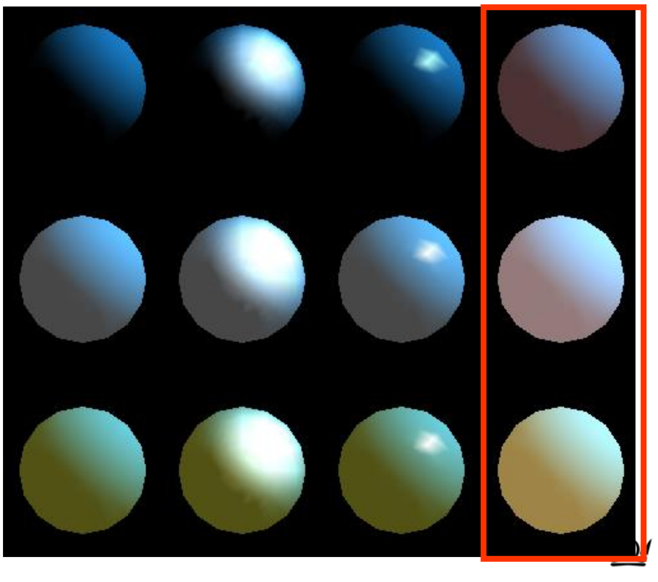


Material properties

glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialf(GL_FRONT, GL_SHININESS, low_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, no_mat);

float mat_ambient_color[] = {0.8, 0.8, 0.2, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_specular[] = {1.0, 1.0, 1.0, 1.0};

Material properties



Material properties

```
glMaterialfv(GL_FRONT, GL_AMBIENT, no_mat);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_emission[] = {0.3, 0.2, 0.2, 1.0};
```

Material properties

```
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

float mat_ambient[] = {0.7, 0.7, 0.7, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_emission[] = {0.3, 0.2, 0.2, 1.0};
```

Material properties

```
glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient_color);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
glMaterialfv(GL_FRONT, GL_SPECULAR, no_mat);
glMaterialf(GL_FRONT, GL_SHININESS, no_shininess);
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);

float mat_ambient_color[] = {0.8, 0.8, 0.2, 1.0};
float mat_diffuse[] = {0.1, 0.5, 0.8, 1.0};
float mat_emission[] = {0.3, 0.2, 0.2, 1.0};
```