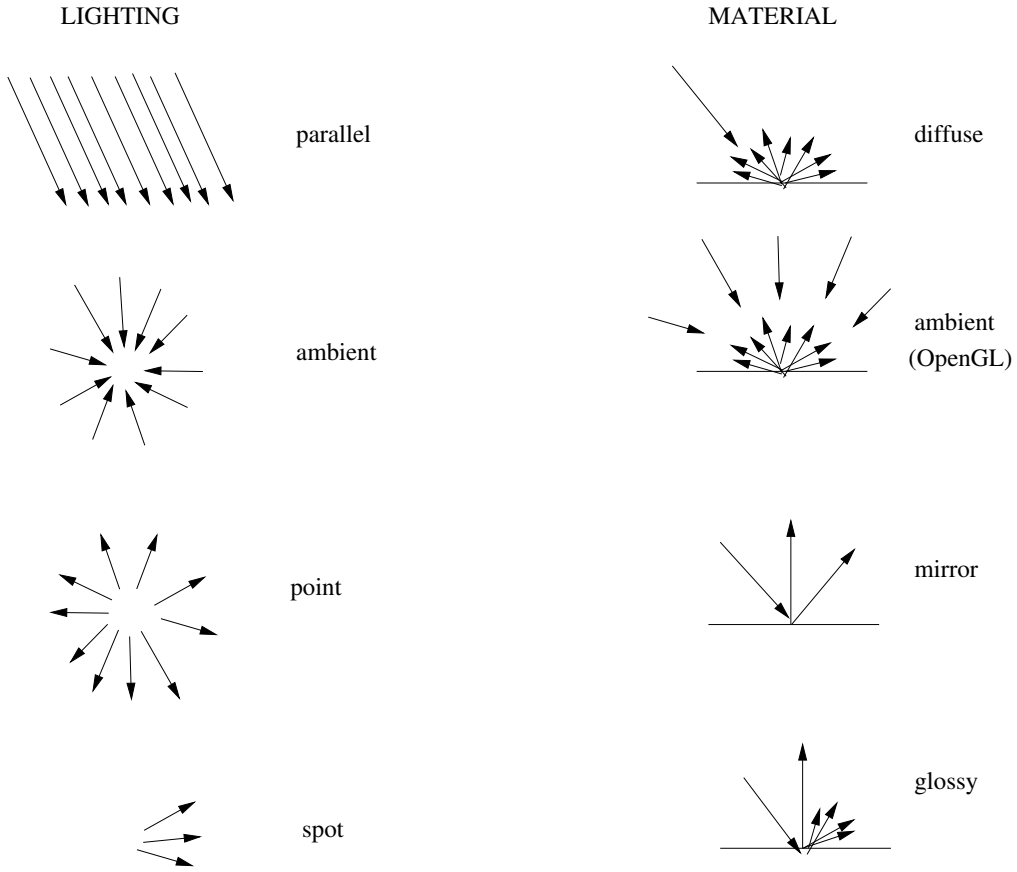
We now move to third part of the course where we will be concerned mostly with what intensity values to put at each pixel in an image. We will begin with a few simple models of lighting and surface reflectance. I discussed these qualitatively, and then gave a more detailed quantitative description of the model.



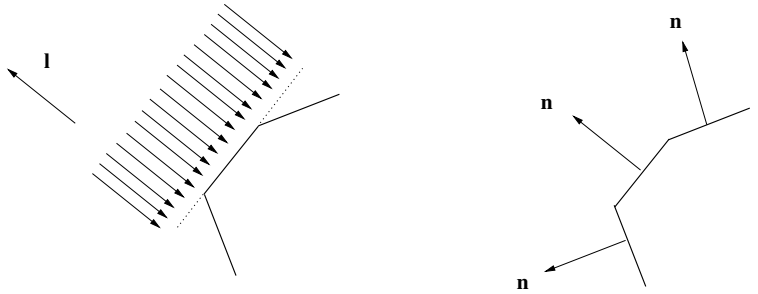
Light sources and illumination

You know what light sources are you probably have not thought about how to model them. Think about the sun vs. a lamp vs. a spotlight vs. the sky. All of these are light sources, but each illuminates the scene in its own way.

Sunlight (Parallel source, "point source at infinity")

Typical lighting models account for the directionality of lighting, namely that light travels along rays. One simple model of directional lighting is the *parallel source*. According to this model, any light ray that arrives at a surface point $\mathbf{x}$ comes from a single direction. We typically say $\mathbf{l}$ is the direction *to* the source and often it is a unit vector. We can think of this as a "sunny day" model. Since the sun is far away, all rays from the sun that reach the earth can be treated as parallel to each other.

The amount of light arriving *per unit surface area* at a point $\mathbf{x}$ and from direction $\mathbf{l}$ depends on the surface normal $\mathbf{n}(\mathbf{x})$. To understand this, think of the parallel rays from the source as having a certain density. The more intense the light source, the greater the density of the rays. The number of rays that reach a unit area patch of surface is proportional to $\mathbf{n}(\mathbf{x}) \cdot \mathbf{l}$, namely to the cosine of the angle between the surface normal and the light source. To understand this, consider the sketch below. The parallel vectors indicate the rays arriving at the surface. There are three surfaces (polygons) of identical size. These are indicated by 1D cross-sections only, but you should imagine they are 2D surface patches with the same area. For the two surfaces that are tilted away from the light source, the number of rays "caught" by the surface is lower than the number caught by the surface that faces the light source directly. The number of rays caught depends on $\mathbf{n}(\mathbf{x}) \cdot \mathbf{l}$.



lightbulb (local point source)

The parallel source is a good model for sunlight but it is not a good model for sources that are positioned *in* the scene. For such lights it is common to use a *local point light source* model. According to this model, light is emitted from a point $\mathbf{x}_l \in \Re^3$ and radiates outwards from that point. I'll first describe a model in which light radiates uniformly out in all directions. Then I'll describe a model in which the directional intensity is not uniform (spotlight). *The order of presentation here differs from in the slides, but covers the same main ideas.*

According to the point source model, the amount of light reaching a surface of unit size depends on three factors:

- the distance between the light source and the surface; If the surface is moved further away from the source, then it will receive less light and hence will become darker. The reason that distant surfaces receive less light is that the light radiating from a source is spread out over a sphere whose area increases with distance squared.

- the fraction of light that the source emits in the direction of the surface: Some sources emit light in certain directions more than others. Car headlights, for example, are designed to throw light on the road ahead, rather than off to the side.

- the surface normal $\mathbf{n}(\mathbf{x})$: If the surface normal points directly at the light source then it will receive more light than if it is oriented obliquely to the source. The reasoning is the same as we argued above for the parallel source;

Consider a point light source located at position $\mathbf{x_l}$ in the scene. A scalar function $I_l(\mathbf{x})$ captures the dependence of the strength of the illumination (the density of rays) at $\mathbf{x}$ as a function of its

position relative to the source. This includes both a distance and direction effect. The distance effect is naturally modelled as a function of distance r between $\mathbf{x}$ and $\mathbf{x}_l$,

$$r \;=\; \| \mathbf{x} - \mathbf{x}_l \| .$$

What is this function? As light radiates from a point source, its energy is spread out on a sphere which has area proportional to r^2. The physics suggests that the intensity should fall off as $\frac{1}{r^2}$, but this assumes the source is a single point. Real sources are not points. Rather, they have a finite volume. This suggests a more general model of the strength of the source as a function of scene position, namely

$$I_l(\mathbf{x}) = \frac{1}{ar^2 + br + c}$$

where r is now the distance to some (arbitrary) point within the source volume. Have constants $c > 0$ (and $a, b > 0$ too) prevents the intensity from going to infinity as $r \to 0$. OpenGL allows such a general falloff function as we will see.

The above function only specifies the strength of the source, but does not specify the direction of the source. To model the directional effect, we define a unit vector $\mathbf{l}(\mathbf{x})$ at each $\mathbf{x}$ which indicates the direction from $\mathbf{x}$ to the point light source $\mathbf{x}_l$:

$$\mathbf{l}(\mathbf{x}) = \frac{\mathbf{x}_l - \mathbf{x}}{\| \mathbf{x}_l - \mathbf{x} \|}.$$

We can then weight this unit vector (direction) with the distance effect:

$$I_l(\mathbf{x})\ \mathbf{l}(\mathbf{x}) \;=\; \frac{1}{(ar^2 + br + c)}\ \frac{\mathbf{x}_l - \mathbf{x}}{\| \mathbf{x}_l - \mathbf{x} \|}$$

which specifies the illumination at any point $\mathbf{x}$.

Spotlight

Many lights in a scene do not illuminate equally in all directions but rather are designed to send most of their illumination in particular directions. A spotlight or car headlight are two examples. Even a typical desklamp is designed to have non-uniform directional properties – the light is supposed to illuminate the desk, rather than shining directly into one's eyes.

To capture this directionality effect, we can modify the scalar $I_l(\mathbf{x})$ of the point source by consider the direction here as well. For example, suppose the light source at $\mathbf{x}_l$ sends out its peak illumination in direction $\mathbf{l}_{\mathbf{peak}}$. One common and simple model for the strength of the source as function of direction (relative ot the peak direction) is as follows:

$$(\mathbf{l}_{\mathbf{peak}} \cdot \frac{\mathbf{x} - \mathbf{x}_\mathbf{l}}{\| \mathbf{x} - \mathbf{x}_\mathbf{l} \|})^{\text{spread}}$$

where the dot product decreases away from the peak direction, and spread controls how fast the falloff is with direction. A laser beam would have a very high spread, whereas a spread close to 0 would have no directional effect. Note that this is *not* a physics-based model. Rather, it is just a simple qualitative model that is meant to capture the fact that some sources are more directional than others.

We can combine this directional effect with the distance effect discussed earlier to have a model of a directional local source such as a spotlight:

$$I_l(\mathbf{x})\ \mathbf{l}(\mathbf{x}) \ = \ \frac{1}{(ar^2+br+c)} \ \ \frac{\mathbf{x}_l - \mathbf{x}}{\| \mathbf{x}_l - \mathbf{x} \|} \ \ (\mathbf{l_{peak}} \cdot \frac{\mathbf{x} - \mathbf{x_l}}{\| \mathbf{x} - \mathbf{x_l} \|})^{\text{spread}}$$

In this model, $\mathbf{l(x)}$ is a unit vector defines at position $\mathbf{x}$ which denotes the direction from which light is coming. The scalar $I_l(\mathbf{x})$ captures the strength of this source. [ASIDE: for those of you with more advanced calculus background, the above is a *vector field*, namely a vector defined at each position $\mathbf{x}$ in space.]

Surface reflectance

The above discussion was concerned with how much light *arrives* at a surface point $\mathbf{x}$ per unit surface area. To model the intensity of light *reflected* from a surface point $\mathbf{x}$, we must specify not just the lighting, but also the surface material. Let's look at the classic models.

Diffuse (matte, Lambertian)

The first model is diffuse (or matte or *Lambertian*[1]) reflectance. This model states that the intensity of light reflected from a surface point $\mathbf{x}$ does not depend on the position from which the point is viewed. This will more clear by the end of the lecture.

For a parallel light source in direction $\mathbf{l}$, the intensity of light reflected from a point $\mathbf{x}$ on a Lambertian surface can be modelled as

$$I_{diffuse}(\mathbf{x}) = I_l \ k_d(\mathbf{x}) \ max(\mathbf{n} \cdot \mathbf{l}, 0)$$

where I_l is the intensity of the source and k_d is the fraction of light reflected by the diffuse component. The max operation is needed because surfaces that face away from the light do not receive direct illumination. The cosine of the angle is negative for such surfaces.

For a local point light source (see top of this page), the intensity of light reflected from a point $\mathbf{x}$ on a Lambertian surface can be modelled as

$$I_{diffuse}(\mathbf{x}) = I_l(\mathbf{x}) \ k_d(\mathbf{x}) \ max(\mathbf{n}(\mathbf{x}) \cdot \mathbf{l}(\mathbf{x}), 0)$$

Ambient light

In real scenes, surfaces that face away from a point light source or a parallel light source still receive some illumination, although this is only indirectly via reflections from other surfaces in the scene. We do not want surfaces that face away from a light source to appear black in our image. To avoid this, we add on a component called the "ambient" component.

$$I_{ambient}(\mathbf{x}) \ = \ I_l k_{ambient}(x).$$

I'll mention more about this later when I discuss OpenGL and in an upcoming lecture (on shadows and interreflections).

[1] Johann Lambert lived in the 18th century.

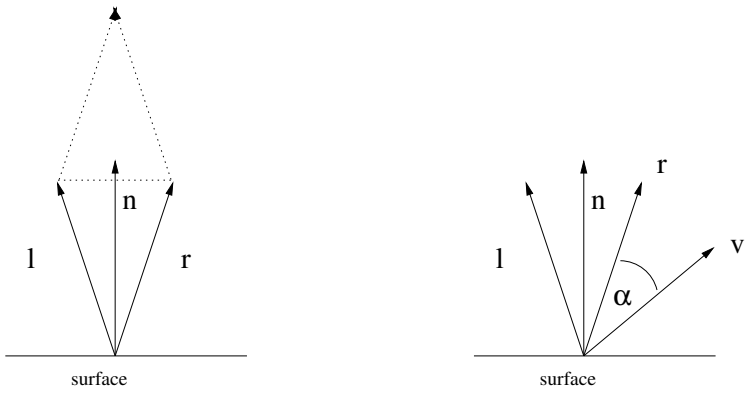
Mirror reflectance

A mirror surface can be modelled as follows. Assume that light source is in direction $\mathbf{l}$ which is a unit vector. To keep the notation simple, we do not distinguish whether the light source is parallel or a point here, and we just write $\mathbf{l}$ rather than $\mathbf{l}(\mathbf{x})$. The light that is reflected from the mirror obeys the rule: "the angle of incidence equals the angle of reflection". Call the unit vector that is the direction of reflection $\mathbf{r}$. The vectors $\mathbf{n}$, $\mathbf{l}$, $\mathbf{r}$ all lie in a plane. Using simple geometry as illustrated in the figure below on the left, one can see that

$$\mathbf{l} + \mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\ \mathbf{n}.$$

Thus,

$$\mathbf{r} = 2(\mathbf{n} \cdot \mathbf{l})\mathbf{n} - \mathbf{l}$$



Glossy/shiny/specular surfaces - the Phong and Blinn-Phong models

In between Lambertian surfaces and mirror surfaces are those that we would call (more or less) shiny. Examples include polished surfaces such as floors and apples, oily surfaces such as skin, and plastic surfaces. Such surfaces do not obey the Lambertian reflectance model because the intensity of the light reflected from a surface point depends on where the camera is placed, i.e. from where the surface point is viewed. An example is the "highlight" on an apple or floor tile. As you move, the location of the highlight slides along the surface.

Shiny surfaces are not perfect mirrors. Rather, the light reflected from these surfaces is concentrated in a *cone* of directions containing the mirror direction $\mathbf{r}$. The reflected light will be brightest if it is viewed along the ray in the mirror reflection direction. The intensity will fall off continuously if it viewed along other rays that begin at $\mathbf{x}$ but have slightly different reflection direction.

One popular model of shiny surfaces that roughly captures the above phenomena is the *Phong lighting model*. Phong's model takes the Lambertian model and adds a shiny component, also called *specular* component. For a given surface point, the intensity of the specular component depends on how "close" the mirror reflection direction $\mathbf{r}$ is to the viewing direction $\mathbf{v}$, that is, the direction from $\mathbf{x}$ to the camera. (Notice that, for the first time, we need to specify explicitly where the camera is. The unit vector $\mathbf{v}$ points from the surface point to the camera.) See the figure above right.

Phong's model considers angular distance between the unit vectors $\mathbf{r}$ and $\mathbf{v}$. It says that the intensity of the specular component of the light reflected in direction $\mathbf{v}$ is proportional to the cosine of the angle between $\mathbf{r}$ and $\mathbf{v}$ and raises this value to some power, which we can think of as the "shininess". I write the glossy component of the Phong lighting model:

$$I_{specular}(\mathbf{x}) \;=\; I_s \; k_s \; (\mathbf{r} \cdot \mathbf{v})^e.$$

where

- I_s is a scalar that describes the intensity of the light source. In terms of the physics, this should be the same scalar as was used in the diffuse reflection component model, since the lighting is not affected when you swap from a diffuse to a shiny surface! [ASIDE: As we will see later, OpenGL allows you to use different lighting properties for the diffuse versus glossy components. This makes no physical sense but graphics people aren't bothered by it.]
- k_s is proportional to the fraction of light reflected in the glossy/specular manner;
- the exponent e is the sharpness of the specular component. e.g. If e is large (100 or more) then the surface will be more mirror-like, whereas if e is small, then the specular reflection will be spread out over a wide range of directions. Note that this is a similar mathematical model to what is used for spotlights. Again, it is *not* a physics-based model, but rather just a qualitative model that describes a directional effect.

Note that each of the above $I_s, k_s, \mathbf{r}$, and $\mathbf{v}$ can depend on $\mathbf{x}$.

Consider a point $\mathbf{x}$ on the surface such that

$$\mathbf{v}(\mathbf{x}) \approx \mathbf{r}(\mathbf{x}),$$

that is, the mirror reflection direction is about the same as the direction to the camera. According to the Phong model, such points produce a peak intensity value since $\mathbf{v} \cdot \mathbf{r} \approx 1$. Such intensity peaks are called *highlights*. You can find highlights on many shiny surfaces, such as polished fruit (apples), ceramic pots, plastic toys, etc.

OpenGL uses a slightly different model of specular reflection. Define the *halfway* vector at a surface point:

$$\mathbf{H} = \frac{\mathbf{l} + \mathbf{v}}{|\, \mathbf{l} + \mathbf{v} \,|}$$

where $\mathbf{l}$ is the direction of the lighting at this point. If $\mathbf{H} = \mathbf{n}$, then $\mathbf{r}$ will be the mirror reflection direction and the peak of the highlight should appear at the surface point. As $\mathbf{n}$ deviates from $\mathbf{H}$, the mirror reflection direction $\mathbf{r}$ will deviate more from the viewer direction $\mathbf{v}$. The Blinn-Phong is similar to the Phong model and can be written,

$$I_{specular}(\mathbf{x}) \;=\; I_s \; k_s \; (\mathbf{H} \cdot \mathbf{n})^{\text{shininess}}.$$

See Exercises for the advantage of Blinn-Phong.

Lighting and Material in OpenGL

In the lecture slides, I gave an overview of how light sources and materials are defined in OpenGL. In a nutshell, you can define light sources which can be parallel, or uniform local sources, or spotlights. You define a light with

```
glLightf( light, parameterName, parameter )
```

and the parameter names of the light are:

```
GL_AMBIENT                      GL_SPOT_DIRECTION
GL_DIFFUSE                      GL_SPOT_EXPONENT
GL_SPECULAR                     GL_SPOT_CUTOFF
GL_POSITION                     GL_ CONSTANT_ATTENUATION
                                GL_LINEAR_ATTENUATION
                                GL_QUADRATIC_ATTENUATION
```

OpenGL allows you to define the RGB color of the light as values between 0 and 1. Weirdly, you can define different colors for the ambient, diffuse, and specular components. I say it is weirdly because lights should be independent of the surfaces they illuminate. It is the surfaces that are diffuse or specular, not the lights. Anyhow, what about the surfaces?

Surfaces are defined by:

```
glMaterialfv( face, parameterName, parameters )
```

where `face` specifies whether the front or back (or both) are drawn, and the parameter names are as show below and `___` is a RGB or RGBA tuple. (I'll explain A later in the course).

```
glMaterial(GL_FRONT, GL_AMBIENT, ___ )
glMaterial(GL_FRONT, GL_DIFFUSE, ___ )
glMaterial(GL_FRONT, GL_SPECULAR, ___ )
glMaterial(GL_FRONT, GL_SHININESS, ___)
```

We can relate the above to the models presented earlier. In the first part of the lecture, we talked about $I(\mathbf{x})$ but really we had one of these functions for each of RGB. In OpenGL, the RGB values are determined by :

$$I(\mathbf{x}) = I_{diffuse}(\mathbf{x}) + I_{specular}(\mathbf{x}) + I_{ambient}(\mathbf{x})$$

where the range is between 0 and 1.

OpenGL does not allow for mirror surfaces, in the general sense I discussed above, although it does provide something more specific, namely environment mapping. I'll describe what this is in an upcoming lecture.

Finally, in the slides, I briefly discussed material modelling beyond OpenGL. I mentioned BRDF's (bidirectional reflection distribution functions), subsurface scattering, etc. Do have a look, though I don't expect you to know this in any detail since I only showed pictures but didn't go into the models.

Shading polygons

At the end of lecture 6, I sketched out how to rasterize a polygon. One of the steps there was to choose the colors at each point inside the polygon. The most straightforward way to fill a polygon is to consider, for each pixel (x, y) in that polygon, what is the surface point (x, y, z) in the scene that projects to that pixel. If we knew the surface normal at this scene point and the lighting and the material of the surface at this scene point, then we could choose the intensity based on the models we discussed above. While this approach sounds straightforward, there are subtleties that arise. There are also several shortcuts that we can take to make the choice of colors more efficient.

Surface normals on polygons: OpenGL

In OpenGL, if you define a polygon but you don't specify the surface normal, then OpenGL will automatically compute the normal by fitting a plane and taking the normal to the plane. It is very useful, however, to explicitly define the surface normals. This can be done at each vertex, for example:

```
glBegin(GL_QUAD)
glNormal3f( _, _, _ )
glVertex3f( _, _, _ )
glNormal3f( _, _, _ )
glVertex3f( _, _, _ )
glNormal3f( _, _, _ )
glVertex3f( _, _, _ )
glNormal3f( _, _, _ )
glVertex3f( _, _, _ )
glEnd()
```

OpenGL would then compute a color at each vertex based on the surface normal and a lighting and material model using what I discussed above. Let's now turn to the question of how to "fill" a polygon, namely choose the colors of pixels in the interior.

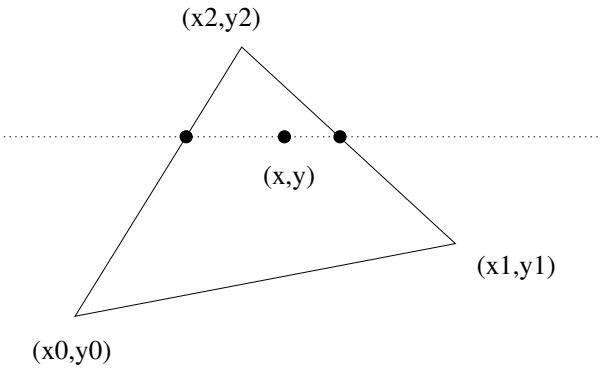
Flat shading

One approach which is perhaps the least interesting is to take one of the vertices of the polygon and choose the color of this particular vertex for all pixels that lie in the polygon. How OpenGL choose one particular vertex is specified here:
`https://www.opengl.org/sdk/docs/man2/xhtml/glShadeModel.xml`
This case is known as *flat* shading. (See `GL_FLAT`).

[ASIDE: In the lecture and slides I said that, for "flat shading", OpenGL computes the normal of the polygon (as defined by the vertices) and uses that normal to compute a single color for polygon. I realized afterwards that this wasn't quite correct, however. In OpenGL, "flat shading" is slightly more general than this. Flat shading means that OpenGL computes the color of one of the vertices of a polygon (depending on the surface normal of that vertex, however it is defined) and uses that vertex color for all of the points in the polygon.]

Linear interpolation (smooth shading)

A more common and interesting approach to choosing the colors in a polygon is to compute the intensities at all of the vertices v of the polygon and then, given these intensities (or more generally RGB values), fill the pixels inside the polygon[2] by *interpolating*.



A simple method of interpolation, which is possible for triangles, is *linear interpolation*. Suppose you have a triangle that projects into the image as shown above, and you have the intensities $I(x_i, y_i)$ for vertices $i = 0, 1, 2$. You wish to interpolate the intensities at the vertices to obtain the intensity $I(x, y)$ at an arbitrary interior point. Consider the edge between image vertices (x_0, y_0) and (x_2, y_2), where $x_0 < x_2$. We can interpolate the intensity along that edge as follows:

$$I(x', y) = I(x_0, y_0)\frac{x_2 - x'}{x_2 - x_0} + I(x_2, y_2)\frac{x' - x_0}{x_2 - x_0}.$$

Equivalently, this can be written:

$$I(x', y) = I(x_0, y_0) + (I(x_2, y_2) - I(x_0, y_0))\frac{x' - x_0}{x_2 - x_0}.$$

Similarly, one linearly interpolates the intensity along the edge between (x_1, y_1) and (x_2, y_2). Finally one linearly interpolates the intensity along the row that corresponds to the dotted line, yielding the desired intensity $I(x, y)$. Again, keep in mind that typically we will be interpolating R, G, and B values, not intensity values.

One can show (see Exercises) that the intensity values $I(x, y)$ that one obtains are a planar fit to the 3D points $(x_0, y_0, I(x_0, y_0))$, $(x_1, y_1, I(x_1, y_1))$, $(x_2, y_2, I(x_2, y_2))$. These are not physical 3D points, of course, they are just points defining a function in $(x, y) \times [0, 1]$, where intensity is in the interval $[0, 1]$.

Linear interpolation is a common method of shading in OpenGL. It is called smooth shading (`GL_SMOOTH`) and it is the default parameter of `glShadeModel`. See the URL on the previous page for the specification, if you are interested.

[2]that is, inside the image projection of the polygon

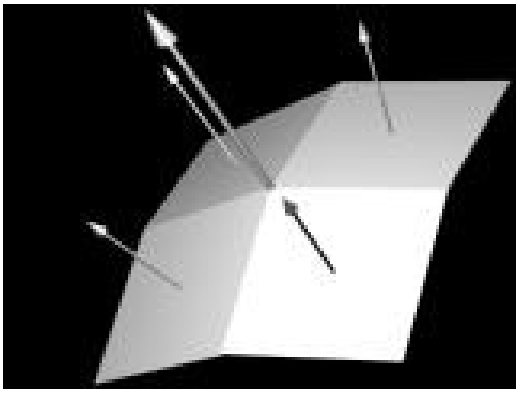
Gouraud shading

If we have a polygon *mesh* and the surface normal is constant across each polygon in the mesh, then we get sharp changes in intensity across the boundary between polygons when the surface normals of neighboring polygons are different. This is fine if the mesh is supposed to have discontinuities at the face boundaries, for example a cube. However, often meshes are used to approximate smooth surfaces. In that case, we don't want sharp intensity edges across face boundaries. Rather we want intensities to be smooth across face boundaries. How can we achieve this?

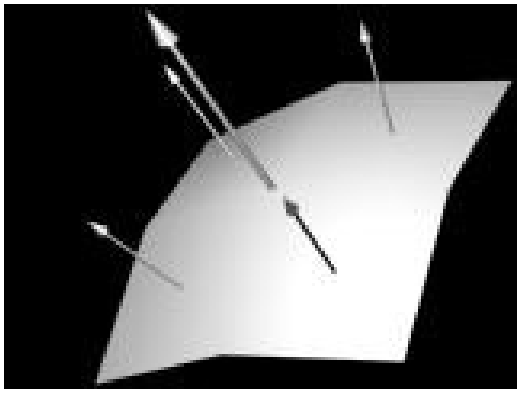
One idea is to define the normal at a vertex to be the average of plane normals $\mathbf{n}_i$ of all faces i that share this vertex,

$$\mathbf{n} \equiv \sum_i \mathbf{n}_i \; / \; \| \sum_i \mathbf{n}_i \| \; .$$

When we define normals in this way, faces that share a vertex will have the same normal at these vertices (since there is just one normal per vertex). When we interpolate the intensities along an edge that is shared by two faces, the intensities will be shared as well since the interpolation will be the same for the two faces along that edge. Thus, when we define normals in this way and we interpolate the intensities within the triangle faces from the intensities at the vertex points (and edges), we will get continuous intensities across edges. This method is called *Gouraud shading* (which is named after its inventor).



flat shading



Gouraud shading

Phong shading (interpolating normals)

The above example suggests that we should relax our concept of what we mean by the surface normal at each point. Taking this idea a step further, we can consider interpolating the surface normals across each polygon. This is called *Phong shading*. The term "Phong shading" is often used interchangeably to describe both Phong's lighting model (mentioned earlier) and the trick described here, where we interpolate the normals. Note, however, that that these are two separate ideas. Phong shading doesn't necessarily use the Phong lighting model. It just means that you interpolate the normals (and use the interpolated normal when coloring an interior point).

What does Phong shading buy you? Suppose you have a shiny smooth surface such an apple which you have represented using a coarse polygonal mesh, with vertices interpolated from the faces as described above. Suppose that the peak of the highlight occurs at a *non-vertex* point. If you use Gouraud shading, then you will miss this highlight peak, and the apple surface will not appear as shiny as it should. If, however, you interpolate the normal, then you have a good chance that some pixel in the interior will have an (interpolated) normal that produces a peak in the highlight.