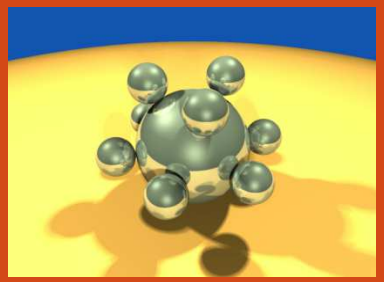
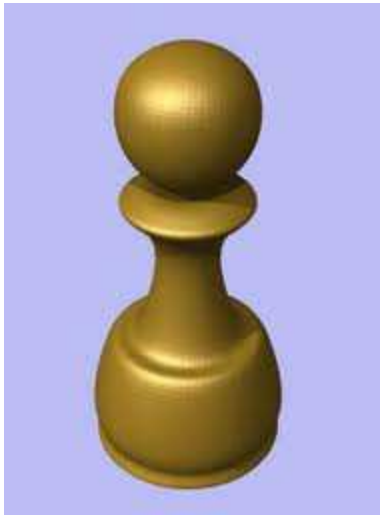


Flat and Smooth Shading

Shading Models: Calculating the Color for Each Screen Pixel

- For some objects, we want to see the individual faces of the polygon (e.g., the barn);
  - we attach ***the same normal*** to all its vertices.

- For others, (eg: sphere or chess pawn) we want to see the surface that the faces approximate and we want the visibility of the edges to be suppressed.
  - we attach ***at each vertex,*** the ***normal*** to the underlying surface

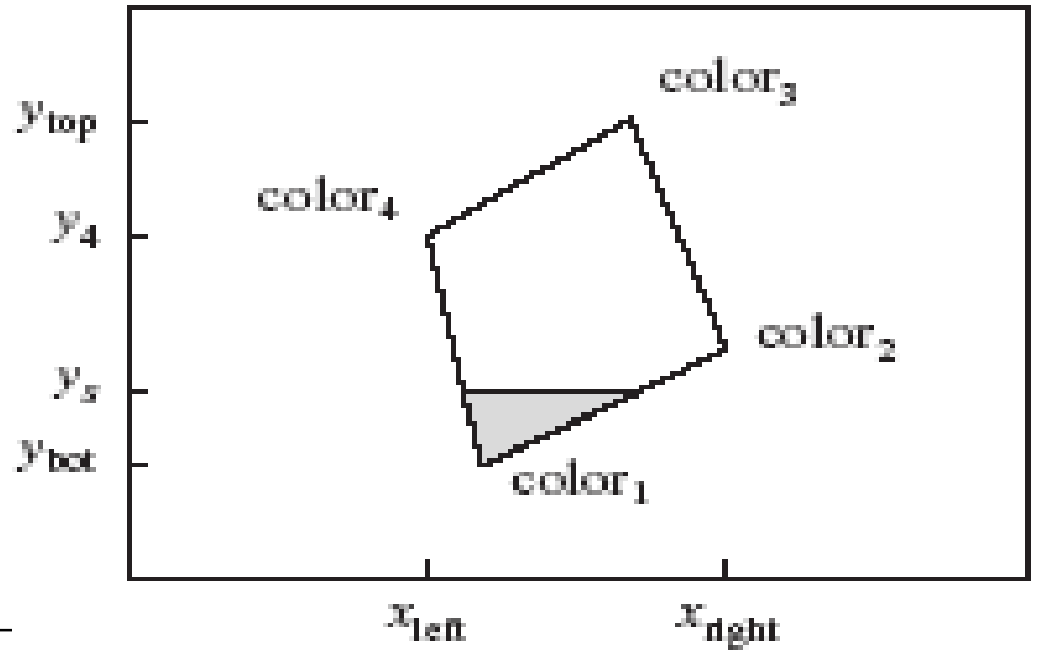


Shading Models (2)

- The normal vectors determine the shading type.
  - **Flat shading** emphasizes individual polygons (barn).
  - **Smooth (Gouraud or Phong) shading** – blends the faces to de-emphasize the edges between them.
- For both kinds of shading,
  1. The vertices are passed down the graphics pipeline,
  2. Shading calculations attach a color to each vertex,
  3. The vertices are converted to screen coordinates and
  4. The face is painted pixel by pixel with the appropriate color.

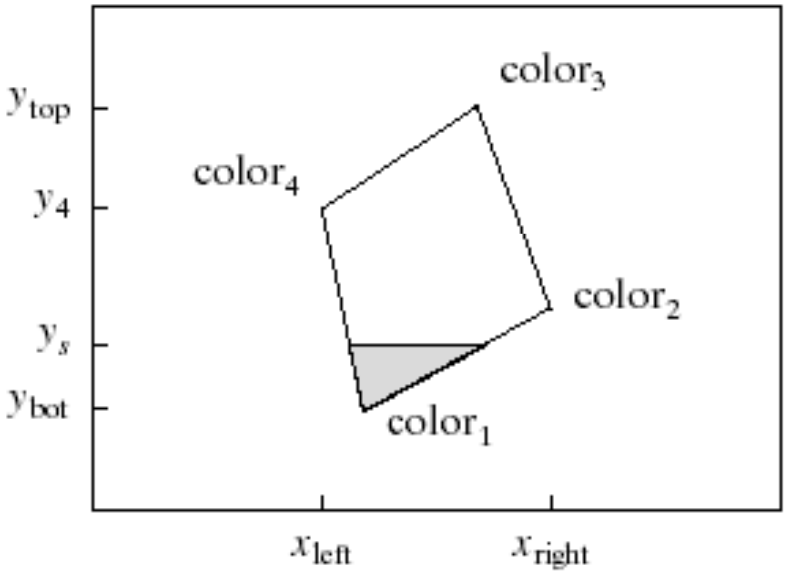
Painting a Face: Polygon Fill

- The face is painted using polygon fill routine called a **tiler** – coloring pixel by pixel.
- For a convex polygon, screen pixels are colored from bottom to top and left to right across the polygon.
- We assume the polygons are *convex*.



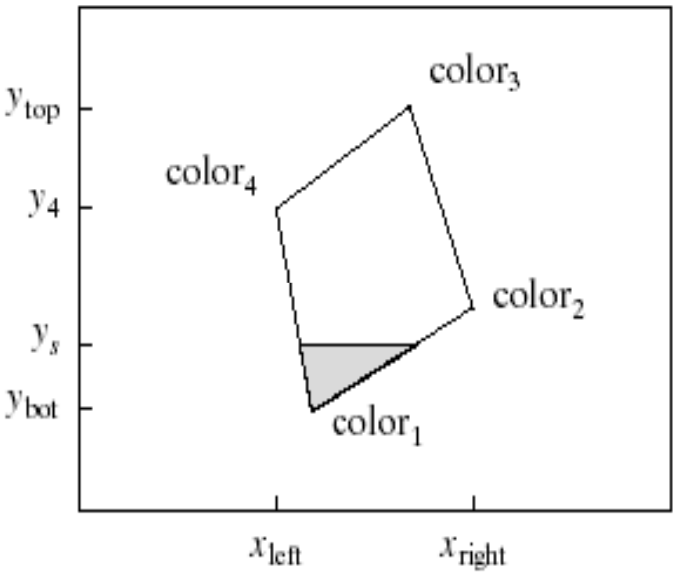
Painting a Face: Polygon Fill

- Filling only convex polygons can be made highly efficient, since at each scan-line there is a single unbroken run of pixels inside the polygon.
- Most implementations of OpenGL exploit this and always fill convex polygons correctly, but do not guarantee to fill non-convex polygons properly.
- The figure shows an example where the face is a convex quadrilateral.



Example

- The screen coordinates of each vertex are noted. The lowest and highest points on the face are y_{bott} and y_{top}, respectively.
- The tiler first fills in the row at $y = y_{bott}$ (in this case a single pixel), then the one at $y_{bott} + 1$, etc.
- At each scan-line, say y_s in the figure, there is a leftmost pixel, x_{left}, and a rightmost pixel, x_{right}.
- The tiler moves from x_{left} to x_{right}, placing the desired color in each pixel. So the tiler is implemented as a simple double loop, looking in pseudo code like



Example (2): Tiler Operation

```
for (int y = ybott; y <= ytop; y++)      // for each scan-line
{
   // find xleft and xright
  for (int x = xleft; x <= xright; x++)// for each relevant
   pixel across this scan-line
   {
         //  find the color c for this pixel
         //  put c into the pixel at (x, y)
   }
}
```

- Hidden surface removal is also easily accomplished within this double loop.

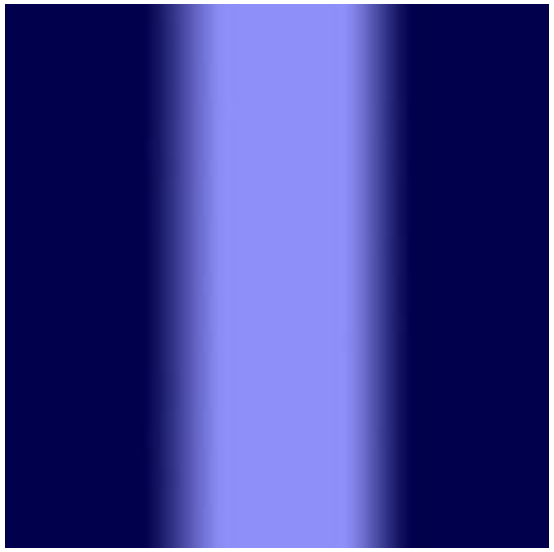
Flat Shading

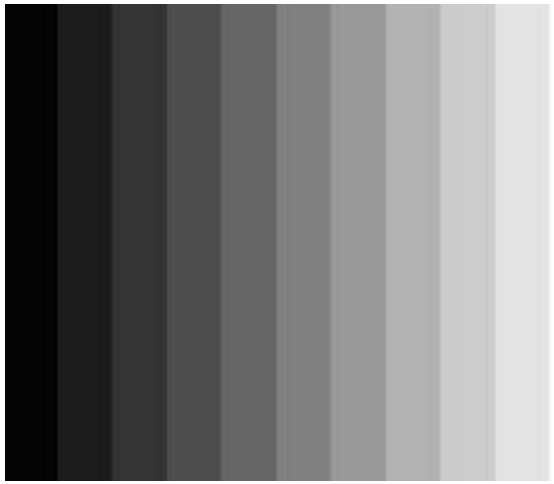
- We must calculate the correct color c for each pixel from the vertex colors.
- Flat Shading: Use the same color for every pixel in a face - usually the color of the first vertex.
- When the face is flat the diffuse light component changes very little over different points.
- GL implements this type of coloring using

glShadeModel(GL_FLAT);

Flat Shading (2)

- Edges appear more pronounced than they would on a real object because of a phenomenon in the eye known as lateral inhibition:
  - A discontinuity in intensity across an object causes the eye to manufacture a Mach band at the discontinuity and a vivid edge is seen. (named after the discoverer, Ernst Mach);



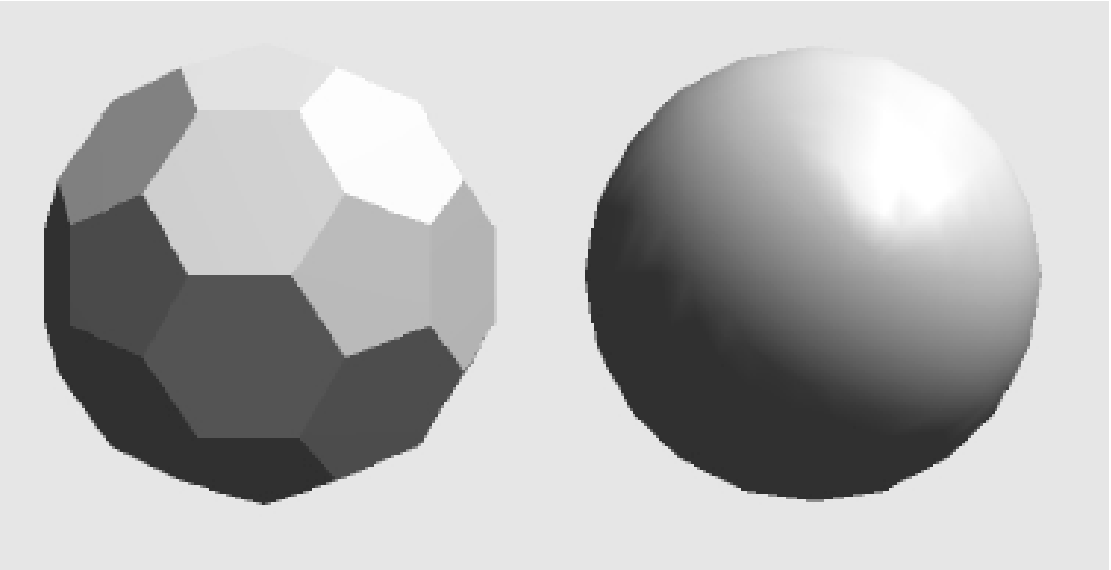


Flat Shading (3)

- Specular highlights are rendered poorly with flat shading:
  - If there happens to be a large specular component at the representative vertex, that brightness is drawn uniformly over the entire face.
  - If a specular highlight doesn't fall on the representative point, it is missed entirely.
- Consequently, we usually do not include the specular reflection component in the shading computation.

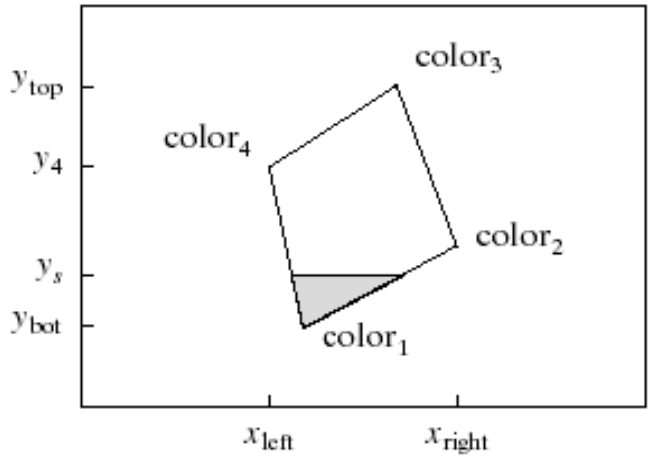
Smooth Shading

- Smooth (Gouraud or Phong) Shading
- Tries to de-emphasize edges between faces.



Gouraud Shading

- Gouraud shading uses linear interpolation of colors between vertices:
- Computes a different value of c for each pixel.
- Color at left edge (or right edge) is linear interpolation between colors at top and bottom of left (right) edge (interpolation in y).



- Top has color c4 and bottom has color c1:

- $color_{left}$ = lerp (c1, c4, f), where,f is the fraction of the way that y_s lies between y_4 and y_{bot}

Gouraud Shading (2)

- Likewise, the $color_{right}$ at the right edge is a linear interpolation of the top and bottom colors of the right edge.
- Color at pixel x along a scanline is a linear interpolation of colors between $color_{left}$ and $color_{right}$ edges.
- Calculations must be made separately for R, G, and B.

$$c(x) = lerp(color_{left}, color_{right}, \frac{x - x_{left}}{x_{right} - x_{left}})$$

- To increase efficiency the color at pixel x+1 is computed incrementally.

$$c(x+1) = c(x) + \frac{color_{right} - color_{left}}{x_{right} - x_{left}}$$

Gouraud Shading (3)

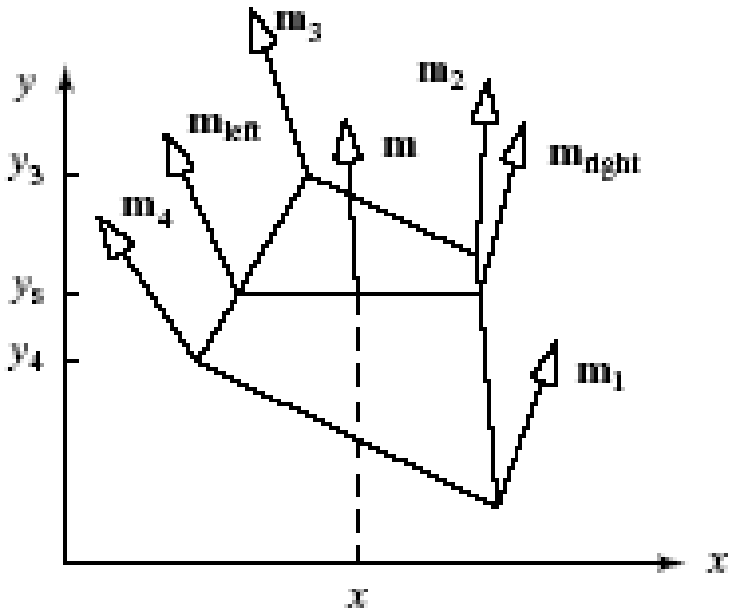
- Gouraud shading is more expensive than flat shading
- OpenGL implements this coloring with glShadeModel(GL_SMOOTH);

Gouraud Shading (4)

- Because colors are formed by interpolating rather than computing colors at every pixel, Gouraud shading does not render highlights well.
- Consequently, we usually do not include the specular reflection component in the shading computation.

Phong Shading

- We find the normal vector at each point on the face and we apply the shading model there to find the color
- Compute the normal at each pixel by interpolating the normal at the vertices of the polygon.



- m_{left} and m_{right} are calculated by linear interpolation of m3,m4 and m1,m2.
- m_{left} and m_{right} are interpolated to calculate normal at each x along the scanline.
- This vector, once normalized is used in shading calculation to form the color at that pixel.

Phong Shading (2)

- Since the direction of the normal vector varies smoothly from point to point and more closely approximates that of an underlying smooth surface, **specular highlights are rendered well.**
- Very good, but **time-consuming**! Takes 6 to 8 times longer than Gouraud shading.
- Not implemented by OpenGL.

Comparison of Phong and Gouraud Shading

- Gouraud (left) and Phong (right): notice presence of specular reflection in Phong version.