# Shading Polygons: Flat Shading

Illumination equations are evaluated at surface locations

- so where do we apply them?

## Flat Shading

- do it once per polygon
- fill every pixel covered by polygon with the resulting color

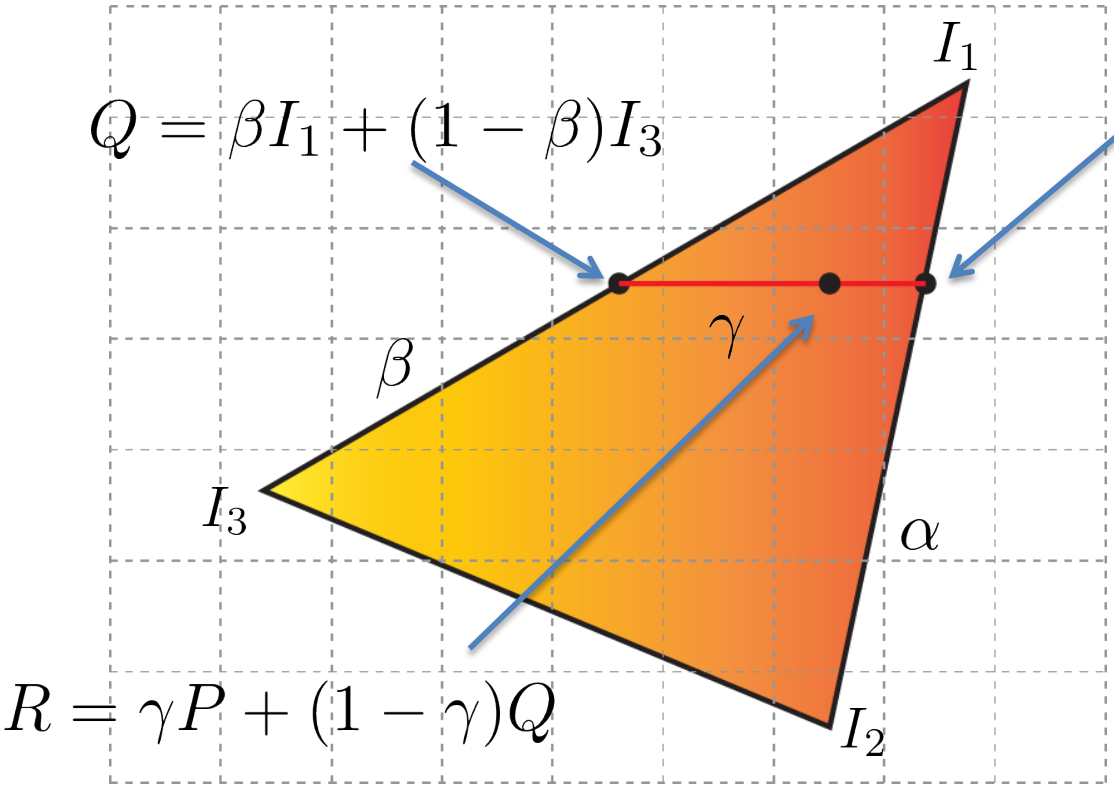
[www.dma.ufg.ac.at/app/link/Grundlagen%3A3D-Grafik/module/9728]

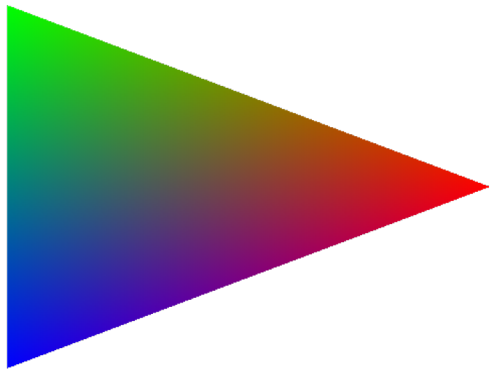
OpenGL — `glShadeModel(GL_FLAT)`

# Shading Polygons: Gouraud Shading

Alternatively, we do lighting calculation once for each vertex

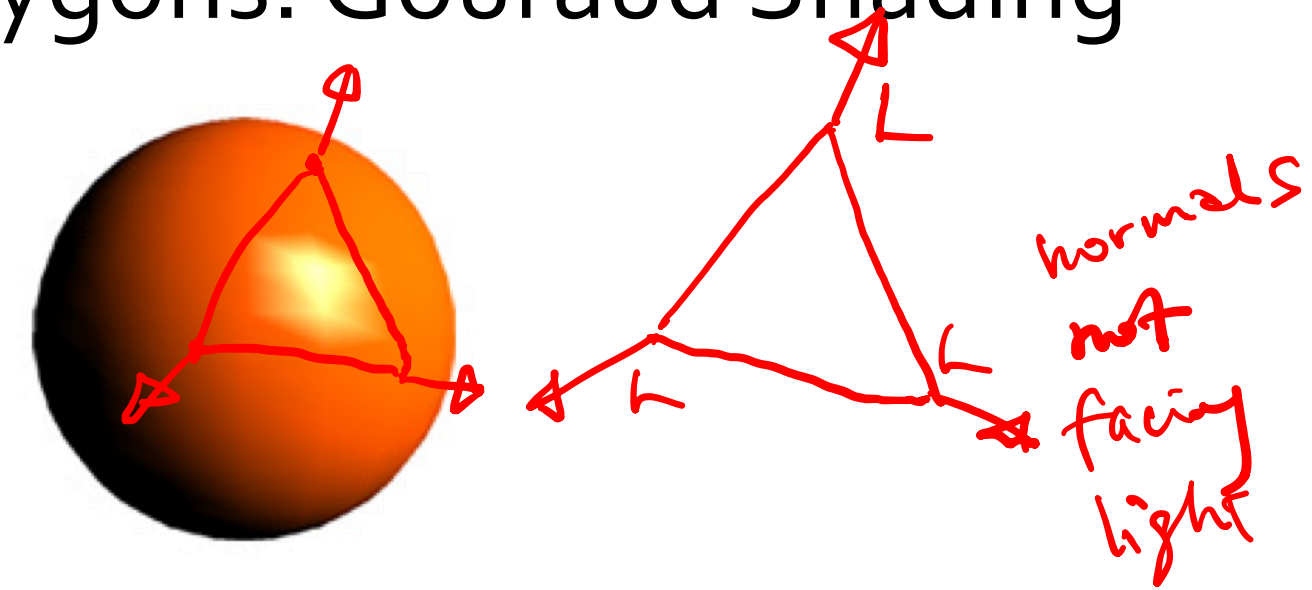
- compute color for each covered pixel
- linearly interpolate colors over polygon



Example of linear interpolation of three different colors at three vertices

# Shading Polygons: Gouraud Shading




If underlying geometry is too coarse, may lead to shading artifacts

Misses details that don't fall on vertex

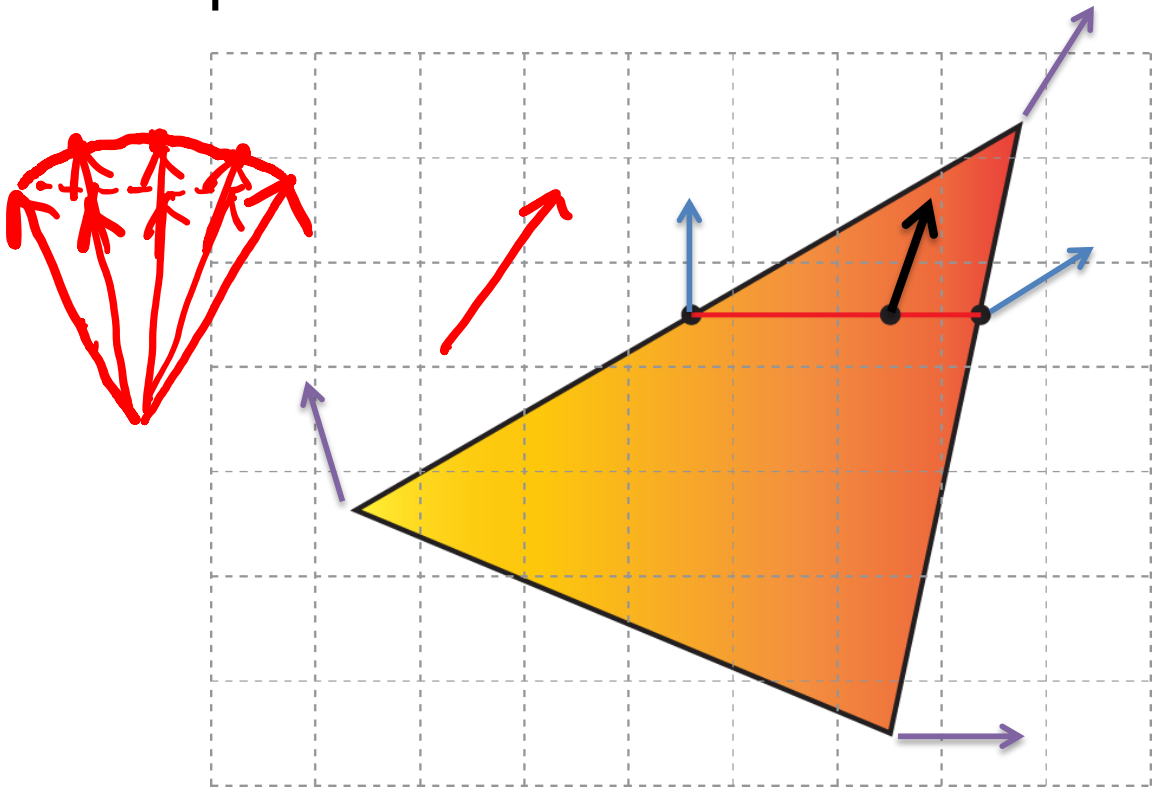
- specular highlights, for instance

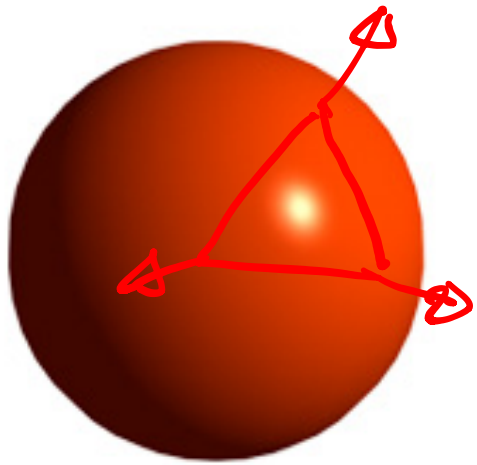
OpenGL — `glShadeModel(GL_SMOOTH)`

# Shading Polygons: Phong Shading

Lighting calculation is carried out for every pixel covered by a triangle.

Surface normal at a pixel is estimated using bilinear interpolation.





Best shading but computationally intensive

OpenGL — not directly supported

# Defining Materials in OpenGL

Just like everything else, there is a current material

- specifies the reflectances of the objects being drawn
- reflectances (e.g., $k_d$) are RGB triples

Set current values with `glMaterial(...)`

```
GLfloat tan1[] = {0.8, 0.7, 0.3, 1.0};
GLfloat tan2[] = {0.4, 0.35, 0.15, 1.0};

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, tan1);
glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, tan1);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, tan2);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50.0);
```

# Defining Lights in OpenGL

A fixed set of lights are available (at least 8)

- turn them on with `glEnable(GL_LIGHTx)`
- set their values with `glLight(…)`

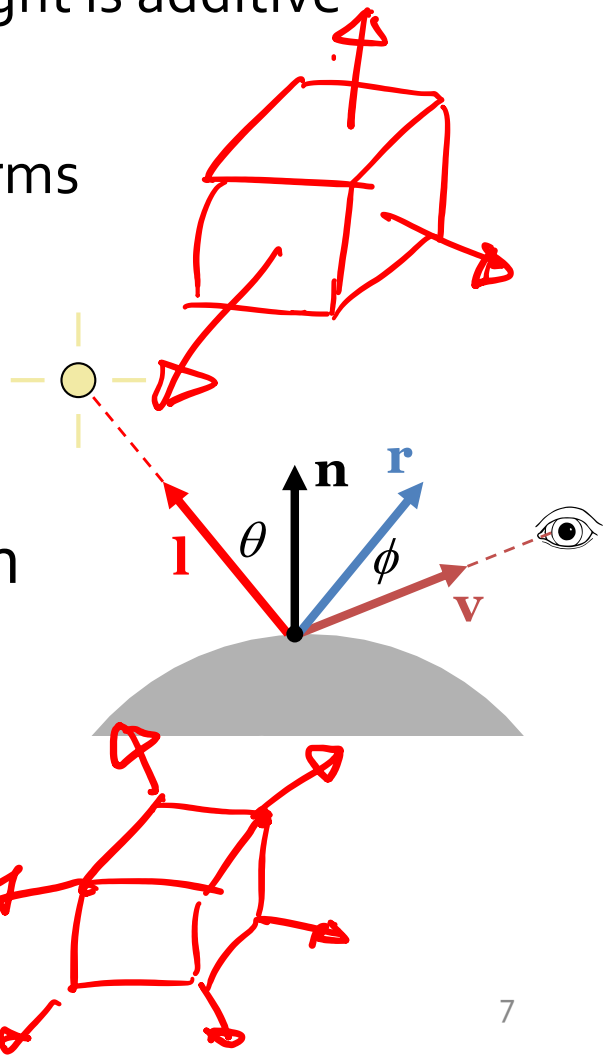
```
GLfloat white[] = {1.0, 1.0, 1.0, 1.0}
GLfloat p[] = {-2.0, -3.0, 10.0, 1.0}; // w=0 for directional light

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

glLightfv(GL_LIGHT0, GL_POSITION, p);
glLightfv(GL_LIGHT0, GL_DIFFUSE, white);
glLightfv(GL_LIGHT0, GL_SPECULAR, white); // can be different

glEnable(GL_NORMALIZE); // guarantee unit normals
```

# Summarizing the Shading Model

We describe local appearance with illumination equations

- consists of a sum of set of components — light is additive
- treat each wavelength independently
- currently: diffuse, specular, and ambient terms

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




Must shade every pixel covered by polygon

- flat shading: constant color
- Gouraud shading: interpolate vertex colors
- Phong shading: interpolate vertex normals