CS 4731: Computer Graphics
Lecture 16: Illumination Models Part 2

Emmanuel Agu

Adding Color

- Sometimes light or surfaces are colored
- Treat R,G and B components separately
- Illumination equation goes from:

$$Illum = ambient + diffuse + specular$$
$$= Ka \times I + Kd \times I \times (N.L) + Ks \times I \times (R.V)^n$$

- To:

$$Illum\_r = Ka\_r \times I\_r + Kd\_r \times I\_r \times (N.L) + Ks\_r \times I\_r \times (R.V)$$
$$Illum\_g = Ka\_g \times I\_g + Kd\_g \times I\_g \times (N.L) + Ks\_g \times I\_g \times (R.V)$$
$$Illum\_b = Ka\_b \times I\_b + Kd\_b \times I\_b \times (N.L) + Ks\_b \times I\_b \times (R.V)$$

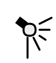
Adding Color

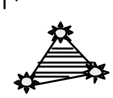
Material	Ambient Kar, Kag,kab	Diffuse Kdr, Kdg,kdb	Specular Ksr, Ksg,ksb	Exponent, n
Black plastic	0.0 0.0 0.0	0.01 0.01 0.01	0.5 0.5 0.5	32
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974
Polished Silver	0.23125 0.23125 0.23125	0.2775 0.2775 0.2775	0.773911 0.773911 0.773911	89.6

Figure 8.17, Hill, courtesy of McReynolds and Blythe

Lighting in OpenGL

- Adopt Phong lighting model
  - specular + diffuse + ambient lights
  - Lighting is computed at vertices
    - Interpolate across surface (Gouraud/smooth shading)
    - Use a constant illumination (get it from one of the vertices)
- Setting up OpenGL Lighting:
  - Light Properties
  - Enable/Disable lighting
  - Surface material properties
  - Provide correct surface normals
  - Light model properties





Light Properties

- Properties:
  - Colors / Position and type / attenuation

glLightfv(light, property, value)

1 2 3

(1) constant: specify which light you want to set the property
E.g: GL_LIGHT0, GL_LIGHT1, GL_LIGHT2 … you can
create multiple lights (OpenGL allows at least 8 lights)
(2) constant: specify which light property you want to set the value
E.g: GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_POSITION
(check the red book for more)
(3) The value you want to set to the property

Property Example

- Define colors and position a light

```
GLfloat light_ambient[] = {0.0, 0.0, 0.0, 1.0};
GLfloat  light_diffuse[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat light_position[] = {0.0, 0.0, 1.0, 1.0};

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

colors

Position

What if I set Position to (0,0,1,0)?

Types of lights

- OpenGL supports two types of lights
  - Local light (point light)
  - Infinite light (directional light)
- Determined by the light positions you provide
  - w = 0: infinite light source (faster)
  - w != 0: point light – position = (x/w, y/w, z/w)

```
GLfloat light_position[] = {x,y,z,w};
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

Turning on the lights

- Turn on the power (for all the lights)
  - glEnable(GL_LIGHTING);
  - glDisable(GL_LIGHTING);

- Flip each light's switch
  - glEnable(GL_LIGHT*n*) (n = 0,1,2,…)

Controlling light position

- Modelview matrix affects a light's position
- Two options:
- Option a:
  - Treat light like vertex
  - Do pushMatrix, translate, rotate, .. **glLightfv position**, popmatrix
  - Then call gluLookat
  - Light moves independently of camera
- Option b:
  - Load identity matrix in modelview matrix
  - Call glLightfv then call gluLookat
  - Light appears at the eye (like a miner's lamp)

Material Properties

- The color and surface properties of a material (dull, shiny, etc)
- How much the surface reflects the incident lights (ambient/diffuse/specular reflecetion coefficients)

  glMaterialfv(face, property, value)

Face: material property for which face (e.g. GL_FRONT, GL_BACK, GL_FRONT_AND_BACK)
Property: what material property you want to set (e.g. GL_AMBIENT, GL_DIFFUSE,GL_SPECULAR, GL_SHININESS, GL_EMISSION, etc)
Value: the value you can to assign to the property

Material Example

- Define ambient/diffuse/specular reflection and shininess

```
GLfloat mat_amb_diff[] = {1.0, 0.5, 0.8, 1.0};    refl. coeff.
GLfloat mat_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat shininess[] = {5.0};          (range: dull 0 – very shiny 128)

glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,
             mat_amb_diff);
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_speacular);
glMaterialfv(GL_FRONT, GL_SHININESS, shininess);
```

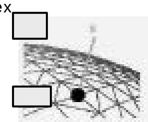
Global light properties

- glLightModelfv(property, value)
- Enable two sided lighting
  - property = GL_LIGHT_MODEL_TWO_SIDE
  - value = GL_TRUE (GL_FALSE if you don't want two sided lighting)
- Global ambient color
  - Property = GL_LIGHT_MODEL_AMBIENT
  - Value = (red, green, blue, 1.0);
- Check the red book for others

Surface Normals

- Correct normals are essential for correct lighting
- Associate a normal to each vertex

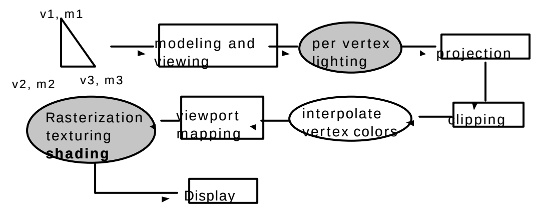
```
glBegin(...)
  glNormal3f(x,y,z)
  glVertex3f(x,y,z)
  ...
glEnd()
```



- The normals you provide need to have a unit length
  - You can use **glEnable (GL_NORMALIZE)** to have OpenGL normalize all the normals

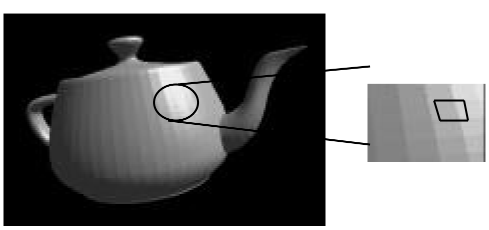
Lighting revisit

- Where is lighting performed in the graphics pipeline?



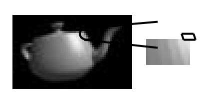
Polygon shading model

- Flat shading - compute lighting once and assign the color to the whole (mesh) polygon



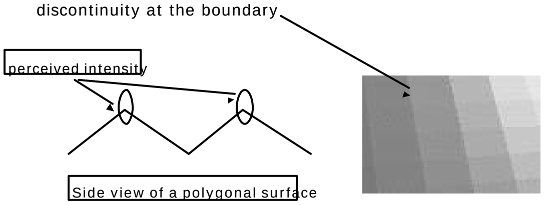
Flat shading

- Only use one vertex normaland material property to compute the color for the polygon
- Benefit: fast to compute
- Used when:
  - Polygon is small enough
  - Light source is far away (why?)
  - Eye is very far away (why?)
- OpenGL command: glShadeModel(GL_FLAT)



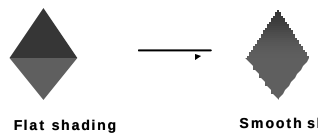
Mach Band Effect

- Flat shading suffers from “mach band effect”
- Mach band effect – human eyes accentuate the discontinuity at the boundary



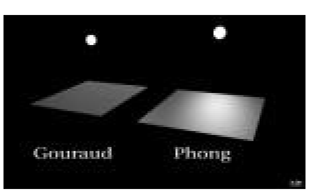
Smooth shading

- Fix the mach band effect – remove edge discontinuity
- Compute lighting for more points on each face



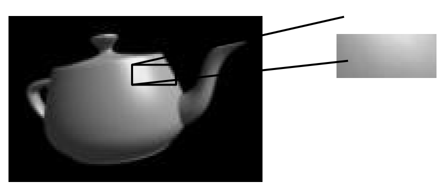
Smooth shading

- Two popular methods:
  - Gouraud shading (used by OpenGL)
  - Phong shading (better specular highlight, not in OpenGL)



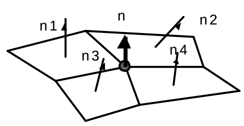
Gouraud Shading

- The smooth shading algorithm used in OpenGL
  glShadeModel(GL_SMOOTH)
- Lighting is calculated for each of the polygon vertices
- Colors are interpolated for interior pixels



Gouraud Shading

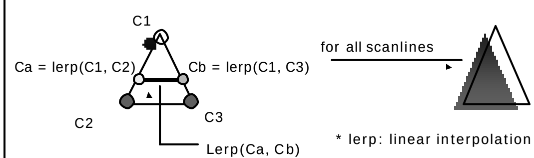
- Per-vertex lighting calculation
- Normal is needed for each vertex
- Per-vertex normal can be computed by averaging the adjust face normals



n = (n1 + n2 + n3 + n4) / 4.0

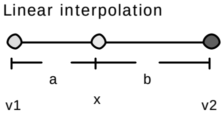
Gouraud Shading

- Compute vertex illumination (color) before the projection transformation
- Shade interior pixels: color interpolation (normals are not needed)



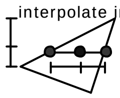
Gouraud Shading

- Linear interpolation



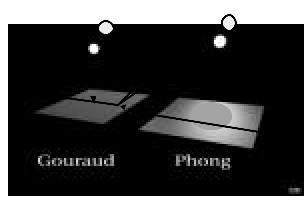
x = a / (a+b) * v1 + b/(a+b) * v2

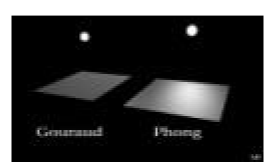
- Interpolate triangle color: use y distance to interpolate the two end points in the scanline, and use x distance to interpolate interior pixel colors



Gouraud Shading Problem

- Lighting in the polygon interior can be inaccurate



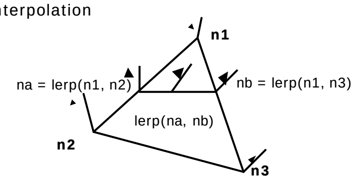


Phong Shading

- Instead of interpolation, we calculate lighting for each pixel inside the polygon (per pixel lighting)
- Need normals for all the pixels – not provided by user
- Phong shading algorithm interpolates the normals and compute lighting during rasterization (need to map the normal back to world or eye space though)

Phong Shading

- Normal interpolation



- Slow – not supported by OpenGL and most graphics hardware

References

- Hill, chapter 8