University of British Columbia
CPSC 314 Computer Graphics
Jan-Apr 2010

Tamara Munzner

Lighting/Shading I

Week 6, Fri Feb 12

http://www.ugrad.cs.ubc.ca/~cs314/Vjan2010

Correction: W2V vs. V2W

slide 38 week3.day3 (Fri Jan 22)

- M_{W2V}=TR $\quad T=\begin{bmatrix}1&0&0&e_x\\0&1&0&e_y\\0&0&1&e_z\\0&0&0&1\end{bmatrix} \quad R=\begin{bmatrix}u_x&v_x&w_x&0\\u_y&v_y&w_y&0\\u_z&v_z&w_z&0\\0&0&0&1\end{bmatrix}$
- we derived position of camera in world
  - invert for world with respect to camera
- $M_{V2W}=(M_{W2V})^{-1}=R^{-1}T^{-1}$

$$M_{view2world}=\begin{bmatrix}u_x&u_y&u_z&0\\v_x&v_y&v_z&0\\w_x&w_y&w_z&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&-e_x\\0&1&0&-e_y\\0&0&1&-e_z\\0&0&0&1\end{bmatrix}=\begin{bmatrix}u_x&u_y&u_z&-\mathbf{e}\bullet\mathbf{u}\\v_x&v_y&v_z&-\mathbf{e}\bullet\mathbf{v}\\w_x&w_y&w_z&-\mathbf{e}\bullet\mathbf{w}\\0&0&0&1\end{bmatrix}$$

Correction: W2V vs. V2W

- $M_{V2W}=(M_{W2V})^{-1}=R^{-1}T^{-1}$

$$M_{view2world}=\begin{bmatrix}u_x&u_y&u_z&0\\v_x&v_y&v_z&0\\w_x&w_y&w_z&0\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&-e_x\\0&1&0&-e_y\\0&0&1&-e_z\\0&0&0&1\end{bmatrix}=\begin{bmatrix}u_x&u_y&u_z&-\mathbf{e}\bullet\mathbf{u}\\v_x&v_y&v_z&-\mathbf{e}\bullet\mathbf{v}\\w_x&w_y&w_z&-\mathbf{e}\bullet\mathbf{w}\\0&0&0&1\end{bmatrix}$$

$$M_{V2W}=\begin{bmatrix}u_x&u_y&u_z&-e_x*u_x+-e_y*u_y+-e_z*u_z\\v_x&v_y&v_z&-e_x*v_x+-e_y*v_y+-e_z*v_z\\w_x&w_y&w_z&-e_x*w_x+-e_y*w_y+-e_z*w_z\\0&0&0&1\end{bmatrix}$$

Correction: Perspective Derivation

slide 30 week4.day3 (Fri Jan 29) z axis flip!

$$\begin{bmatrix}x'\\y'\\z'\\w'\end{bmatrix}=\begin{bmatrix}E&0&A&0\\0&F&B&0\\0&0&C&D\\0&0&-1&0\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

$x'=Ex+Az$
$y'=Fy+Bz$
$z'=Cz+D$
$w'=-z$

$x=left \rightarrow x'/w'=-1$
$x=right \rightarrow x'/w'=1$
$y=top \rightarrow y'/w'=1$
$y=bottom \rightarrow y'/w'=-1$
$z=-near \rightarrow z'/w'=1$
$z=-far \rightarrow z'/w'=-1$

$$y'=Fy+Bz,\quad \frac{y'}{w'}=\frac{Fy+Bz}{w'},\quad 1=\frac{Fy+Bz}{w'},\quad 1=\frac{Fy+Bz}{-z},$$

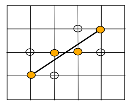
$$1=F\frac{y}{-z}+B\frac{z}{-z},\quad 1=F\frac{y}{-z}-B,\quad 1=F\frac{top}{-(-near)}-B,$$

$$1=F\frac{top}{near}-B$$

News

- P2 due date extended to Tue Mar 2 5pm
  - V2W correction affects Q1 and thus cascades to Q4-Q7
  - perspective correction affects Q8
- TA office hours in lab for P2/H2 questions Fri 2-4 (Garrett)

PPT Fix: Basic Line Drawing



$y=mx+b$

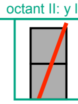
$$y=\frac{(y_1-y_0)}{(x_1-x_0)}(x-x_0)+y_0$$

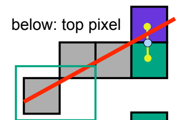
- goals
  - integer coordinates
  - thinnest line with no gaps
- assume
  - $x_0<x_1$, slope $0<\frac{dy}{dx}<1$
  - one octant, other cases symmetric
- how can we do this more quickly?

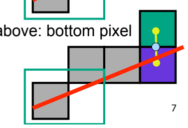
```
Line ( x0, y0, x1, y1)
begin
float dx, dy, x, y, slope ;
dx <= x1 - x0;
dy <= y1 - y0;
slope <= dy/dx;
y <= y0
for x from x0 to x1 do
  begin
    PlotPixel ( x, Round (y) ) ;
    y <= y + slope ;
  end ;
end ;
```

Clarification/Correction II: Midpoint

- we're moving horizontally along x direction (first octant)
  - only two choices: draw at current y value, or move up vertically to y+1?
    - check if midpoint between two possible pixel centers above or below line
  - candidates
    - top pixel: (x+1,y+1)
    - bottom pixel: (x+1, y)
  - midpoint: (x+1, y+.5)
- check if midpoint above or below line
  - below: pick top pixel
  - above: pick bottom pixel
- other octants: different tests
  - octant II: y loop, check x left/right

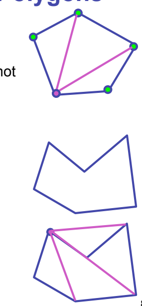






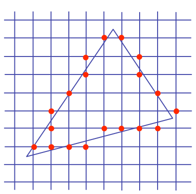
Review: Triangulating Polygons

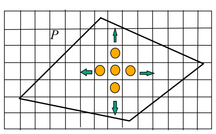
- simple convex polygons
  - trivial to break into triangles
  - pick one vertex, draw lines to all others not immediately adjacent
  - OpenGL supports automatically
    - glBegin(GL_POLYGON) ... glEnd()
- concave or non-simple polygons
  - more effort to break into triangles
  - simple approach may not work
  - OpenGL can support at extra cost
    - gluNewTess(), gluTessCallback(), ...



Review: Flood Fill

- simple algorithm
  - draw edges of polygon
  - use flood-fill to draw interior





PPT Fix: Flood Fill

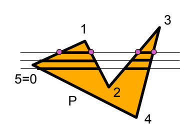
- draw edges
- run:

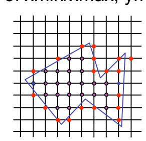
```
FloodFill(Polygon P, int x, int y, Color C)
if not (OnBoundary(x,y,P) or Colored(x,y,C))
begin
    PlotPixel(x,y,C);
    FloodFill(P,x+1,y,C);
    FloodFill(P,x,y+1,C);
    FloodFill(P,x,y-1,C);
    FloodFill(P,x-1,y,C);
end ;
```

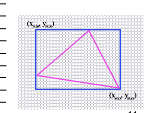
- drawbacks?

Review: Scanline Algorithms

- scanline: a line of pixels in an image
  - set pixels inside polygon boundary along horizontal lines one pixel apart vertically
    - parity test: draw pixel if edgecount is odd
    - optimization: only loop over axis-aligned bounding box of xmin/xmax, ymin/ymax

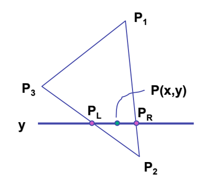






Review: Bilinear Interpolation

- interpolate quantity along L and R edges, as a function of y
  - then interpolate quantity as a function of x

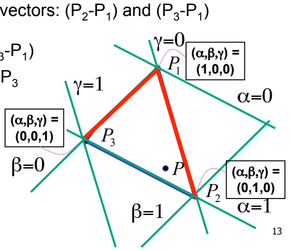


Review: Barycentric Coordinates

- non-orthogonal coordinate system based on triangle itself
  - origin: P_1, basis vectors: (P_2-P_1) and (P_3-P_1)

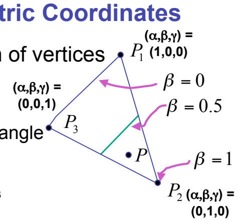
$P = P_1 + \beta(P_2-P_1)+\gamma(P_3-P_1)$
$P = (1-\beta-\gamma)P_1 + \beta P_2+\gamma P_3$
$P = \alpha P_1 + \beta P_2+\gamma P_3$

$\alpha + \beta+ \gamma = 1$
$0 <= \alpha, \beta, \gamma <= 1$



Using Barycentric Coordinates

- weighted combination of vertices
- smooth mixing
- speedup
  - compute once per triangle



$$\begin{cases}P=\alpha\cdot P_1+\beta\cdot P_2+\gamma\cdot P_3\\ \alpha+\beta+\gamma=1\\ 0\le\alpha,\beta,\gamma\le 1 \text{ for points inside triangle}\end{cases}$$

"convex combination of points"

- demo
  http://www.cut-the-knot.org/Curriculum/Geometry/Barycentric.shtml

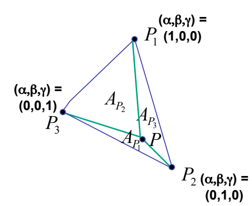
---

Computing Barycentric Coordinates

- 2D triangle area
  - half of parallelogram area
    - from cross product

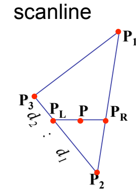
$A = A_{P1} +A_{P2} +A_{P3}$

$\alpha = A_{P1} /A$
$\beta = A_{P2} /A$
$\gamma = A_{P3} /A$



Deriving Barycentric From Bilinear

- from bilinear interpolation of point P on scanline



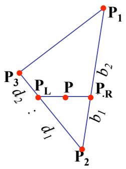
$$P_L=P_2+\frac{d_1}{d_1+d_2}(P_3-P_2)$$

$$=(1-\frac{d_1}{d_1+d_2})P_2+\frac{d_1}{d_1+d_2}P_3=$$

$$=\frac{d_2}{d_1+d_2}P_2+\frac{d_1}{d_1+d_2}P_3$$

Deriving Barycentric From Bilineaer

- similarly

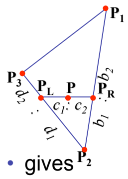


$$P_R = P_2 + \frac{b_1}{b_1+b_2}(P_1 - P_2)$$
$$= (1 - \frac{b_1}{b_1+b_2})P_2 + \frac{b_1}{b_1+b_2}P_1 =$$
$$= \frac{b_2}{b_1+b_2}P_2 + \frac{b_1}{b_1+b_2}P_1$$

17

Deriving Barycentric From Bilinear

- combining



$$P = \frac{c_2}{c_1+c_2} \cdot P_L + \frac{c_1}{c_1+c_2} \cdot P_R$$
$$P_L = \frac{d_2}{d_1+d_2}P_2 + \frac{d_1}{d_1+d_2}P_3$$
$$P_R = \frac{b_2}{b_1+b_2}P_2 + \frac{b_1}{b_1+b_2}P_1$$

- gives

$$P = \frac{c_2}{c_1+c_2}\left(\frac{d_2}{d_1+d_2}P_2 + \frac{d_1}{d_1+d_2}P_3\right) + \frac{c_1}{c_1+c_2}\left(\frac{b_2}{b_1+b_2}P_2 + \frac{b_1}{b_1+b_2}P_1\right)$$

18

Deriving Barycentric From Bilinear

- thus $P = \alpha P_1 + \beta P_2 + \gamma P_3$ with

$$\alpha = \frac{c_1}{c_1+c_2}\frac{b_1}{b_1+b_2}$$
$$\beta = \frac{c_2}{c_1+c_2}\frac{d_2}{d_1+d_2} + \frac{c_1}{c_1+c_2}\frac{b_2}{b_1+b_2}$$
$$\gamma = \frac{c_2}{c_1+c_2}\frac{d_1}{d_1+d_2}$$

- can verify barycentric properties

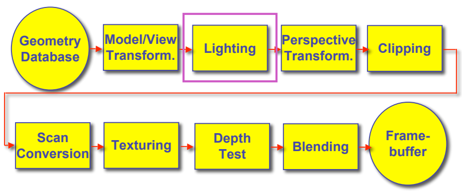
$$\alpha + \beta + \gamma = 1, \qquad 0 \le \alpha, \beta, \gamma \le 1$$

19

Lighting I

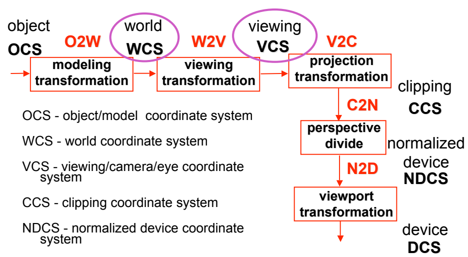
20

Rendering Pipeline



21

Projective Rendering Pipeline



OCS - object/model coordinate system

WCS - world coordinate system

VCS - viewing/camera/eye coordinate system

CCS - clipping coordinate system

NDCS - normalized device coordinate system

DCS - device/display/screen coordinate system

22

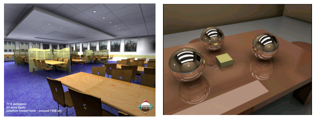
Goal

- simulate interaction of light and objects
- fast: fake it!
  - approximate the look, ignore real physics
- get the physics (more) right
  - BRDFs: Bidirectional Reflection Distribution Functions
- local model: interaction of each object with light
- global model: interaction of objects with each other

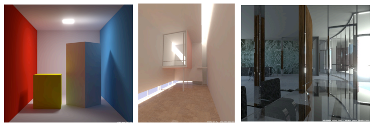
23

Photorealistic Illumination

- transport of energy from light sources to surfaces & points
  - global includes direct and indirect illumination – more later



[electricimage.com]



Henrik Wann Jensen

24

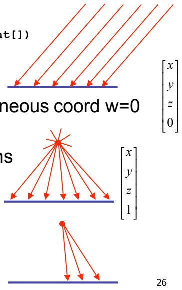
Illumination in the Pipeline

- local illumination
  - only models light arriving directly from light source
  - no interreflections or shadows
    - can be added through tricks, multiple rendering passes
- light sources
  - simple shapes
- materials
  - simple, non-physical reflection models

25

Light Sources

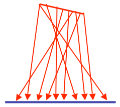
- types of light sources
  - `glLightfv(GL_LIGHT0,GL_POSITION,light[])`
- directional/parallel lights
  - real-life example: sun
  - infinitely far source: homogeneous coord w=0
- point lights
  - same intensity in all directions
- spot lights
  - limited set of directions:
    - point+direction+cutoff angle



26

Light Sources

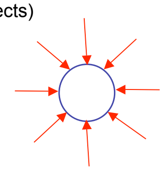
- area lights
- light sources with a finite area
- more realistic model of many light sources
- not available with projective rendering pipeline (i.e., not available with OpenGL)



27

Light Sources

- ambient lights
- no identifiable source or direction
- hack for replacing true global illumination
  - (diffuse interreflection: light bouncing off from other objects)



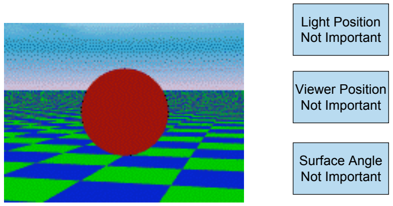
28

Diffuse Interreflection

29

Ambient Light Sources

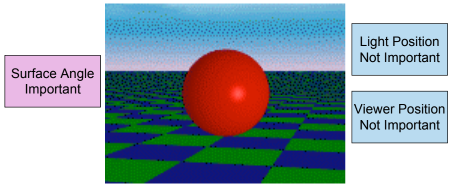
- scene lit only with an ambient light source



30

Directional Light Sources

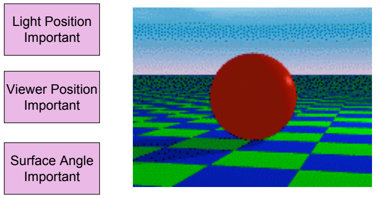
- scene lit with directional and ambient light



31

Point Light Sources

- scene lit with ambient and point light source



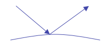
32

Light Sources

- geometry: positions and directions
  - standard: world coordinate system
    - effect: lights fixed wrt world geometry
    - demo: http://www.xmission.com/~nate/tutors.html
  - alternative: camera coordinate system
    - effect: lights attached to camera (car headlights)
  - points and directions undergo normal model/view transformation
- illumination calculations: camera coords

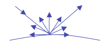
Types of Reflection

- *specular* (a.k.a. *mirror* or *regular*) reflection causes light to propagate without scattering.



- *diffuse* reflection sends light in all directions with equal energy.

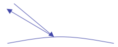
- *mixed* reflection is a weighted combination of specular and diffuse.



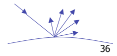
Specular Highlights

Types of Reflection

- *retro-reflection* occurs when incident energy reflects in directions close to the incident direction, for a wide range of incident directions.

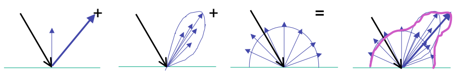


- *gloss* is the property of a material surface that involves mixed reflection and is responsible for the mirror like appearance of rough surfaces.



Reflectance Distribution Model

- most surfaces exhibit complex reflectances
  - vary with incident and reflected directions.
  - model with combination

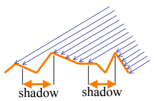


specular + glossy + diffuse = reflectance distribution

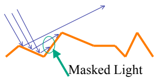
Surface Roughness

- at a microscopic scale, all real surfaces are rough

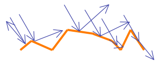
- cast shadows on themselves



- “mask” reflected light:



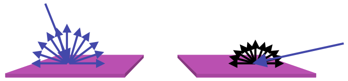
Surface Roughness



- notice another effect of roughness:
  - each “microfacet” is treated as a perfect mirror.
  - incident light reflected in different directions by different facets.
  - end result is mixed reflectance.
    - smoother surfaces are more specular or glossy.
    - random distribution of facet normals results in diffuse reflectance.

Physics of Diffuse Reflection

- ideal diffuse reflection
  - very rough surface at the microscopic level
    - real-world example: chalk
  - microscopic variations mean incoming ray of light equally likely to be reflected in any direction over the hemisphere
  - what does the reflected intensity depend on?

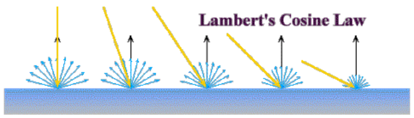


Lambert’s Cosine Law

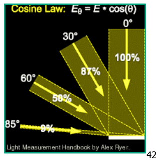
- ideal diffuse surface reflection

  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
- reflected intensity
  - independent of viewing direction
  - depends on surface orientation wrt light
- often called Lambertian surfaces

Lambert’s Law

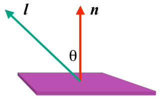


intuitively: cross-sectional area of the “beam” intersecting an element of surface area is smaller for greater angles with the normal.



Computing Diffuse Reflection

- depends on angle of incidence: angle between surface normal and incoming light
  - $I_{diffuse} = k_d I_{light} \cos\theta$
- in practice use vector arithmetic
  - $I_{diffuse} = k_d I_{light} (\mathbf{n} \bullet \mathbf{l})$



- **always normalize vectors used in lighting!!!**
  - **n, l** should be unit vectors
- scalar (B/W intensity) or 3-tuple or 4-tuple (color)
  - k_d: diffuse coefficient, surface color
  - I_{light}: incoming light intensity
  - $I_{diffuse}$: outgoing light intensity (for diffuse reflection)

Diffuse Lighting Examples

- Lambertian sphere from several lighting angles:

- need only consider angles from 0° to 90°
  - *why?*
  - *demo: Brown exploratory on reflection*
  - http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/reflection2D/reflection_2d_java_browser.html