University of
British Columbia

Review 2

Lines and Curves

- Explicit - one coordinate as function of the others

$$y = f(x)$$

$$z = f(x, y)$$

line

$$y = mx + b$$

$$y = \frac{(y_2 - y_1)}{(x_2 - x_1)}(x - x_1) + y_1$$

circle

$$y = \pm\sqrt{r^2 - x^2}$$

University of British Columbia

Lines and Curves

- Parametric – all coordinates as functions of common parameter

$$(x, y) = (f_1(t), f_2(t))$$

$$(x, y, z) = (f_1(u,v), f_2(u,v), f_3(u,v))$$

line

$$x(t) = x_1 + t(x_2 - x_1)$$
$$y(t) = y_1 + t(y_2 - y_1)$$
$$t \in [0,1]$$

circle

$$x(\theta) = r\cos(\theta)$$
$$y(\theta) = r\sin(\theta)$$
$$\theta \in [0, 2\pi]$$

Lines and Curves

- Implicit - define as "zero set" of function of all the parameters

$$\{(x,y):F(x,y)=0\}$$

$$\{(x,y,z):F(x,y,z)=0\}$$

- Defines partition of space

$$\{(x,y):F(x,y)>0\},\{(x,y):F(x,y)=0\},\{(x,y):F(x,y)<0\}$$

Lines and Curves - Implicits

line

$$dy = y_2 - y_1$$

$$dx = x_2 - x_1$$

$$F(x,y) = (x - x_1)\,dy - (y - y_1)\,dx$$

$F(x,y) = 0$ **(x,y) is on line**

$F(x,y) > 0$ **(x,y) is below line**

$F(x,y) < 0$ **(x,y) is above line**

$$F(x,y) = xdy - ydx + (y_1 dx - x_1 dy)$$

circle

$$F(x,y) = x^2 + y^2 - r^2$$

$F(x,y) = 0$ **(x,y) is on circle**

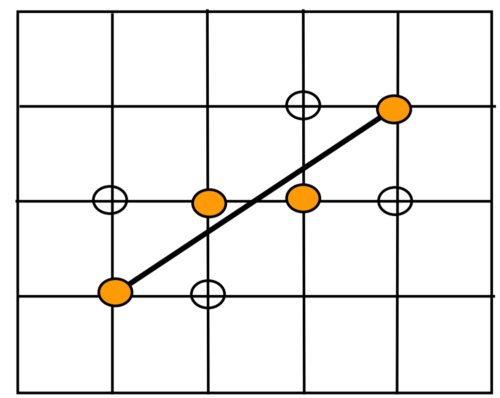
$F(x,y) > 0$ **(x,y) is outside**

$F(x,y) < 0$ **(x,y) is inside**

Basic Line Drawing

Assume $x_1 < x_2$ & line slope absolute value is ≤ 1

$$
\begin{aligned}
&\textbf{Line}\ (x_1, y_1, x_2, y_2)\\
&\text{begin}\\
&\text{float}\ \ dx, dy, x, y, slope\ ;\\
&dx \Leftarrow x_2 - x_1;\\
&dy \Leftarrow y_2 - y_1;\\
&slope \Leftarrow {}^{dy}\!/_{dx}\ ;\\
&y \Leftarrow y_1\\
&\text{for}\ \ x\ \text{from}\ \ x_1\ \text{to}\ \ x_2\ \text{do}\\
&\text{begin}\\
&\qquad \textbf{PlotPixel}\ (x, \textbf{Round}\ (y))\ ;\\
&\qquad y \Leftarrow y + slope\ ;\\
&\text{end}\ ;\\
&\text{end}\ ;
\end{aligned}
$$



Questions:

Can this algorithm use integer arithmetic ?

How do we draw other curves?

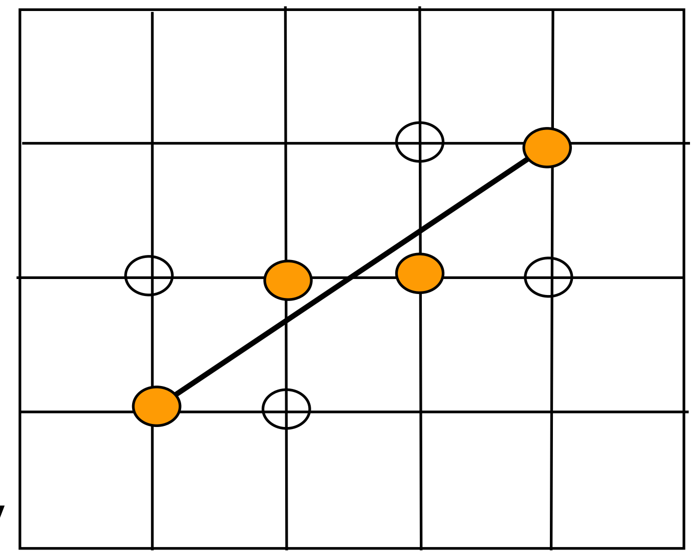
e.g. $y=x^2$ between x_1 and x_2

University of British Columbia

Midpoint (Bresenham) Algorithm

- **Assumptions:**

$$x_2 > x_1, y_2 > y_1 \; and \; \frac{dy}{dx} = \frac{y_2 - y_1}{x_2 - x_1} < 1$$



- **Idea:**
  - Proceed along the line incrementally
  - Have ONLY 2 choices
  - Select one that minimizes error (distance to line)

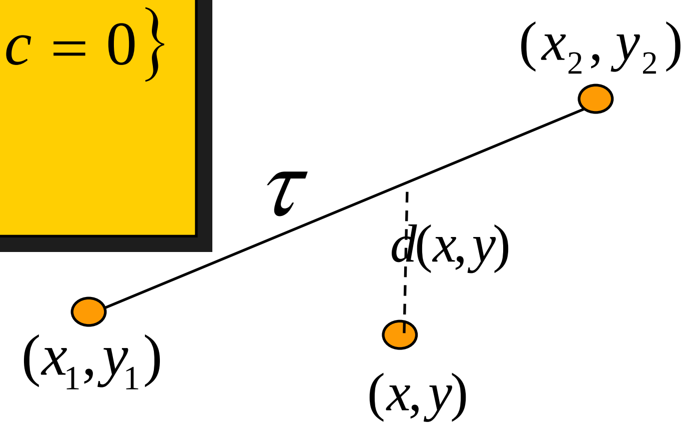
University of
British Columbia

Bresenham Algorithm

Distance (error):

$$\tau = \{(x, y) : ax + by + c = xdy - ydx + c = 0\}$$

$$d(x, y) = 2(xdy - ydx + c)$$

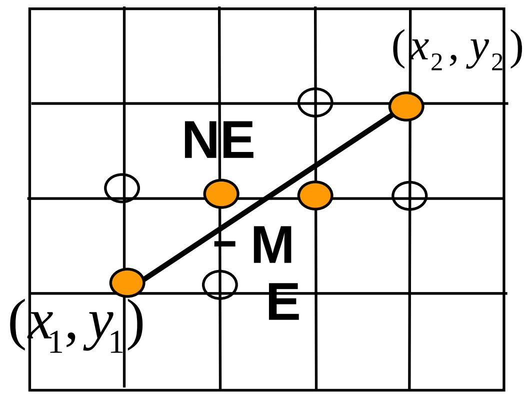


- Given point $P = (x, y)$, $d(x, y)$ is signed distance of *p* to τ (up to normalization factor)
- *d* is zero for $P \in \tau$

University of British Columbia

Midpoint Line Drawing (cont'd)

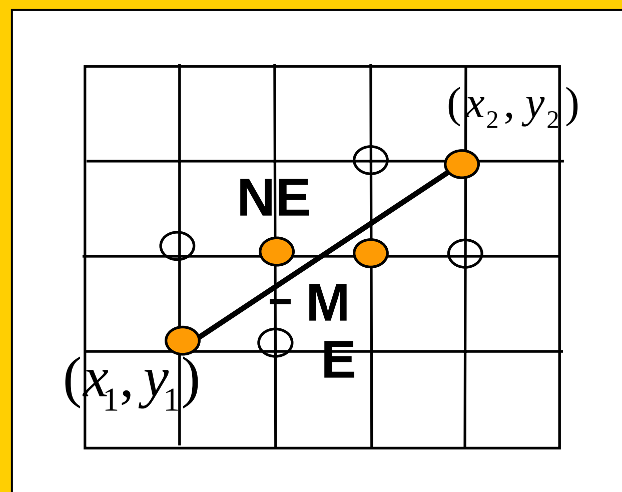
- Starting point satisfies $d(x_1, y_1) = 0$
- Each step moves right (east) or upper right (northeast)
- Sign of $d(x+1, y+\frac{1}{2})$ indicates if to move east or northeast



University of
British Columbia

Midpoint Line Algorithm (version 1)

Line (x_1, y_1, x_2, y_2)
begin
int x, y, dx, dy, d;
$x \Leftarrow x_1$; $y \Leftarrow y_1$;
$dx \Leftarrow x_2 - x_1$; $dy \Leftarrow y_2 - y_1$;
PlotPixel (x, y);
while $(x < x_2)$ do
    $d = (2x + 2)dy - (2y + 1)dx + 2c; // 2((x + 1)dy - (y + .5)dx + c)$
    if $(d < 0)$ then
    begin
        $x \Leftarrow x + 1$;
    end ;
    else begin
        $x \Leftarrow x + 1$;
        $y \Leftarrow y + 1$;
    end ;
    PlotPixel (x, y);
end ;
end ;



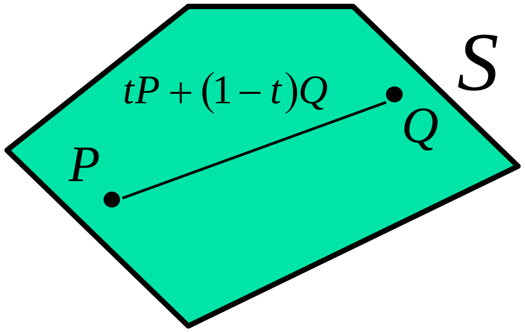

bresenham

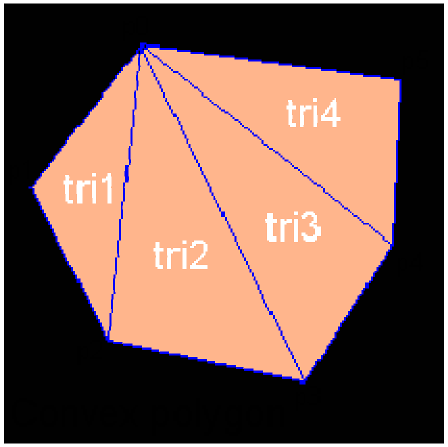
University of British Columbia

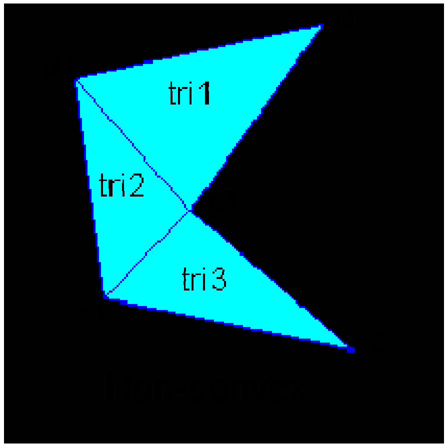
Triangulation

- Convex polygons easily triangulated
- Concave polygons present a challenge
- Convexity - formal definition:

Object S is **convex** iff for any two points $P, Q \in S,\ tP + (1-t)Q \subseteq S,\ t \in [0,1]$.

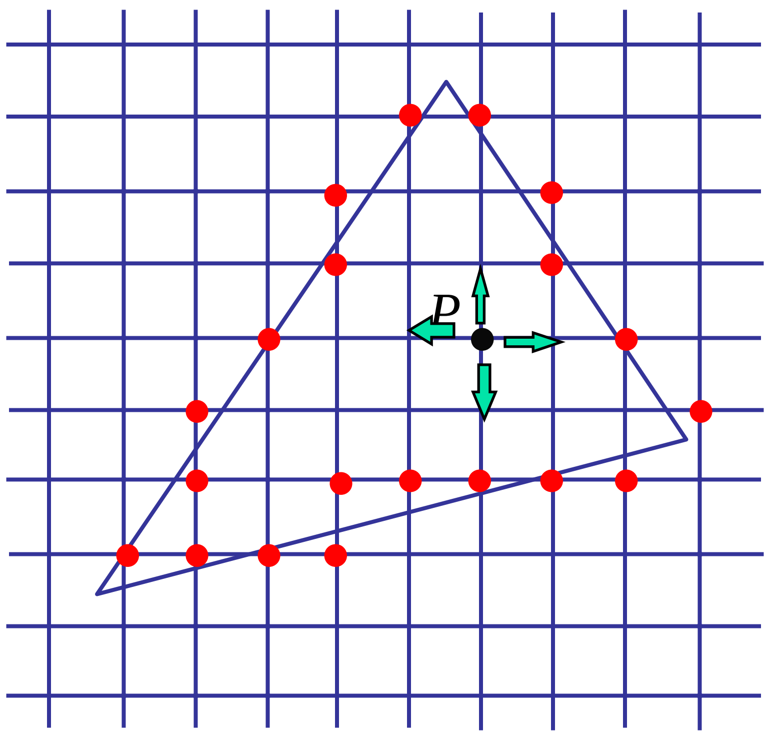






Flood Fill Algorithm

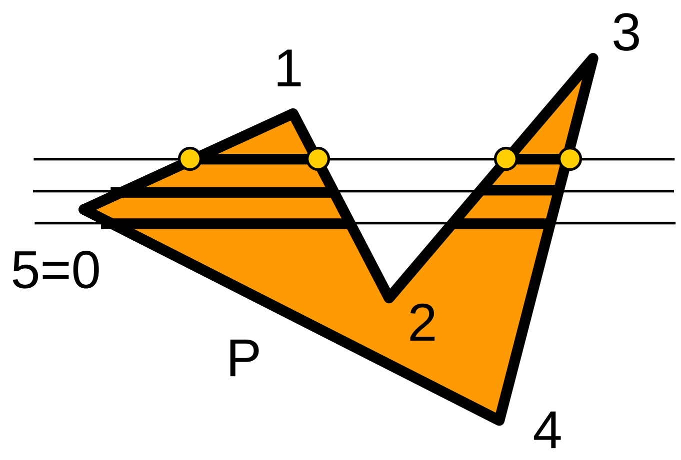
- Input
  - polygon P with rasterized edges
  - $P = (x,y) \in P$ point inside P



University of
British Columbia

Scanline Algorithm

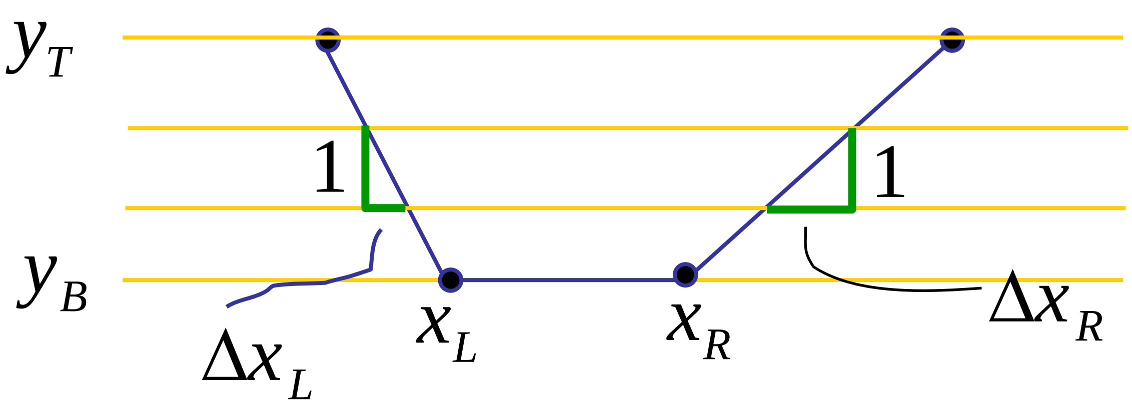
- Observation: Each intersection of straight line with boundary moves it from/into polygon
- Detect (& set) pixels inside polygon boundary (simple closed curve) with set of horizontal lines (pixel apart)



University of
British Columbia

Edge Walking

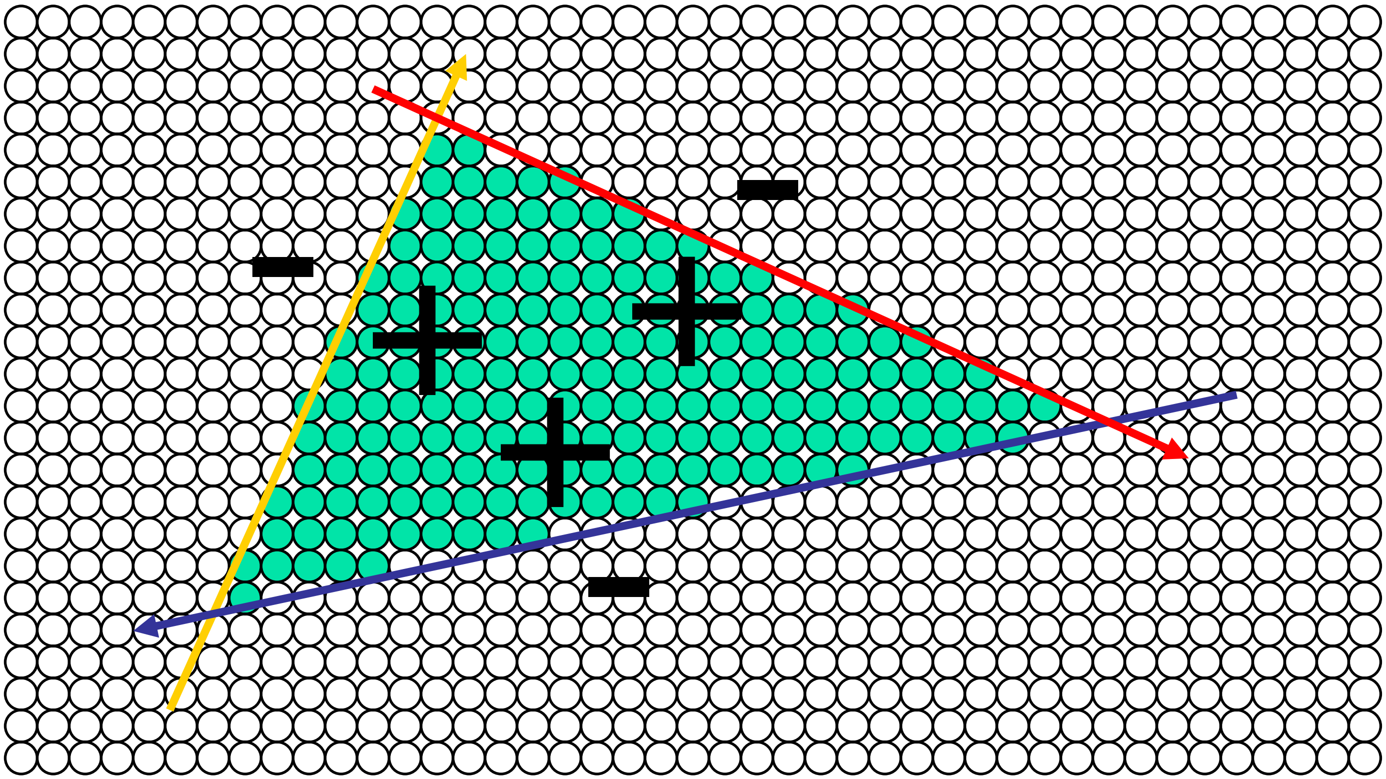
```
for (y=yB; y<=yT; y++) {
  for (x=xL; x<=xR; x++)
    setPixel(x,y);
  xL += DxL;
  xR += DxR;
}
```



University of
British Columbia

Modern Rasterization

- Define a triangle from implicit edge equations:



Barycentric Coordinates

- Area

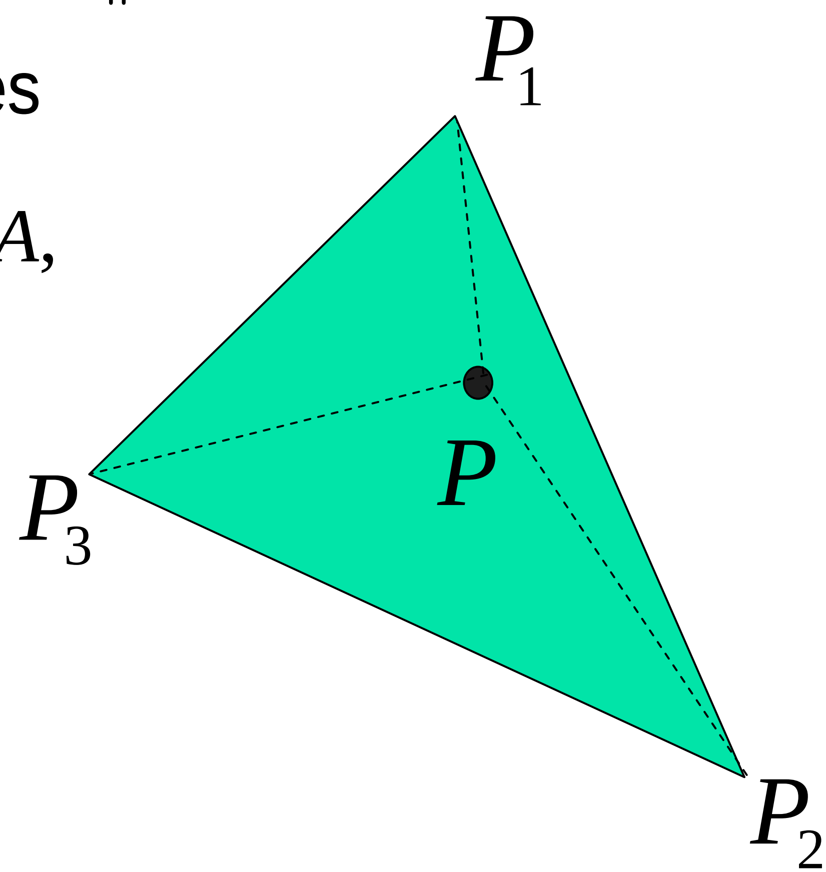
$$A = \frac{1}{2}\left\| \overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3} \right\|$$

- Barycentric coordinates

$$a_1 = A_{P_2P_3P} / A, a_2 = A_{P_3P_1P} / A,$$

$$a_3 = A_{P_1P_2P} / A,$$

$$P = a_1P_1 + a_2P_2 + a_3P_3$$



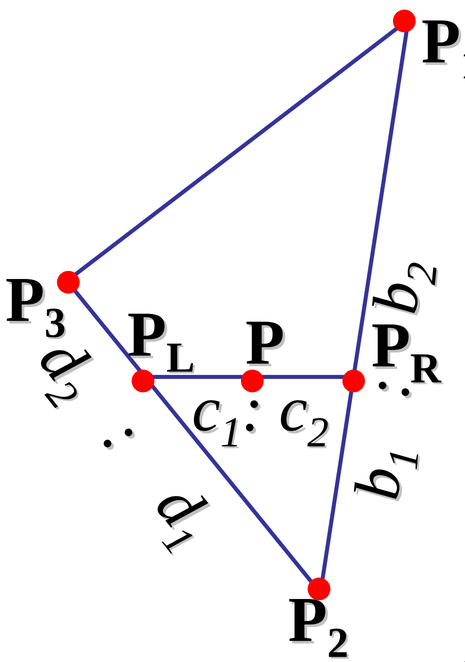
Computing Barycentric Coords

- combining

$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

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

$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$



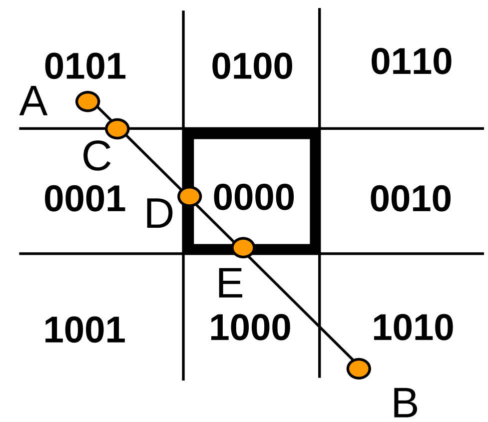
- gives

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

University of British Columbia

Cohen-Sutherland Algorithm (cont'd)

$\textbf{C-S-Clip}(P_0 = (x_0, y_0), P_1 = (x_1, y_1), x_{\min}, x_{\max}, y_{\min}, y_{\max})$
$C_0 \Leftarrow \text{code}(P_0);$ $\quad C_1 \Leftarrow \text{code}(P_1);$
if $((C_0 \text{ and } C_1) \mathrel{!=} 0)$ then return;
if $((C_0 \text{ or } C_1) == 0)$ then draw(P_0, P_1);
else if (OutsideWindow(P_0)) then
begin
$\mathit{Edge} \Leftarrow$ Window boundary of leftmost non-zero bit of C_0;
$P_2 \Leftarrow \overline{P_0, P_1} \cap \mathit{Edge};$
$\textbf{C-S-Clip}(P_2, P_1, x_{\min}, x_{\max}, y_{\min}, y_{\max});$
end
else
$\mathit{Edge} \Leftarrow$ Window boundary of leftmost non-zero bit of C_1;
$P_2 \Leftarrow \overline{P_0, P_1} \cap \mathit{Edge};$
$\textbf{C-S-Clip}(P_0, P_2, x_{\min}, x_{\max}, y_{\min}, y_{\max});$
end



AB → CB → DB → DE

bit	1	0
1	$y < y_{\min}$	$y \geq y_{\min}$
2	$y > y_{\max}$	$y \leq y_{\max}$
3	$x > x_{\max}$	$x \leq x_{\max}$
4	$x < x_{\min}$	$x \geq x_{\min}$

University of British Columbia

C-S Algorithm for convex polygons – full version

$\mathbf{C\text{-}S\text{-}Clip}(\text{poly} = P_0,\ldots,P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$

for i = 1 to n $C_i \Leftarrow \text{code}(P_i)$;

if $((C_0 \text{ and } C_1 \text{ and} \ldots \text{and } C_n) \mathrel{!}= 0)$ then return;

if $((C_0 \text{ or } C_1 \text{ or} \ldots \text{or } C_n) == 0)$ then draw(poly);

else

for i = 1 to n if (OutsideWindow(P_i)) then

begin

$\quad Edge \Leftarrow$ Window boundary of leftmost non-zero bit of C_i;

$\quad P_{i-1,i} \Leftarrow \overline{P_{i-1}, P_i} \cap Edge$; /* if no intersection return P_{i-1} */

$\quad P_{i,i+1} \Leftarrow \overline{P_i, P_{i+1}} \cap Edge$; /* if no intersection return P_{i+1} */

$\quad$ if $(P_{i-1,i} == P_{i-1}$ and $P_{i,i+1} == P_{i+1})$

$\quad\quad \mathbf{C\text{-}S\text{-}Clip}(P_0,\ldots,P_{i-1}, P_{i+1},\ldots,P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$

$\quad$ else if $(P_{i-1,i} == P_{i-1})$ /* no intersection, or exactly on the end-vertex */

$\quad\quad \mathbf{C\text{-}S\text{-}Clip}(P_0,\ldots,P_{i-1}, P_{i,i+1} P_{i+1},\ldots,P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$;

$\quad$ else if $(P_{i,i+1} == P_{i+1})$ /* no intersection, or exactly on the end-vertex */

$\quad\quad \mathbf{C\text{-}S\text{-}Clip}(P_0,\ldots,P_{i-1}, P_{i-1,i}, P_{i+1},\ldots,P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$

$\quad$ else

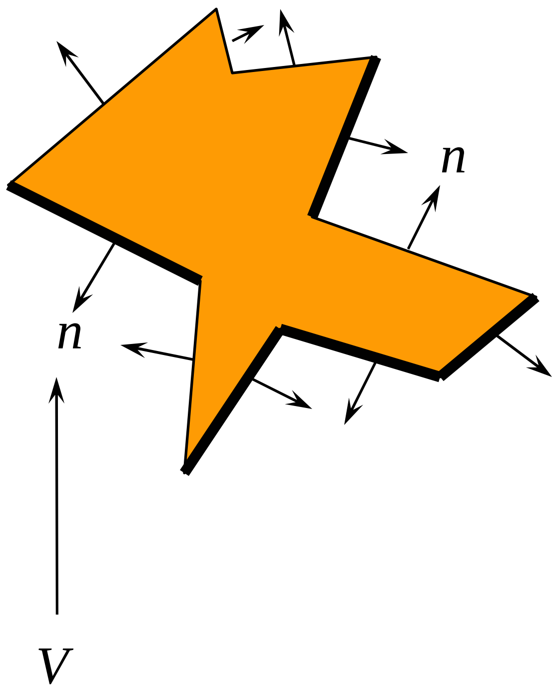
$\quad\quad \mathbf{C\text{-}S\text{-}Clip}(P_0,\ldots,P_{i-1}, P_{i-1,i}, P_{i,i+1}, P_{i+1},\ldots,P_n, x_{\min}, x_{\max}, y_{\min}, y_{\max})$;

end

bit	1	0
1	$y<y_{\min}$	$y\geq y_{\min}$
2	$y>y_{\max}$	$y\leq y_{\max}$
3	$x>x_{\max}$	$x\leq x_{\max}$
4	$x<x_{\min}$	$x\geq x_{\min}$

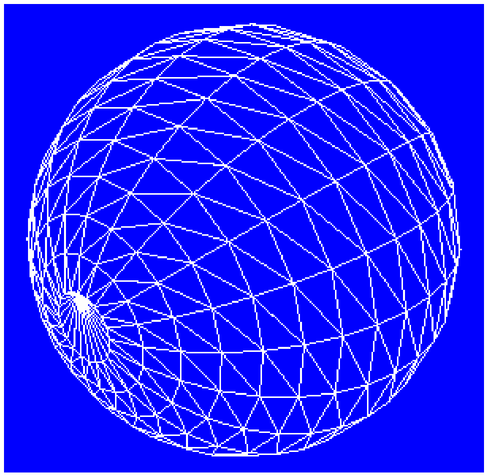
Univers
British C

Back Face Culling (object space)



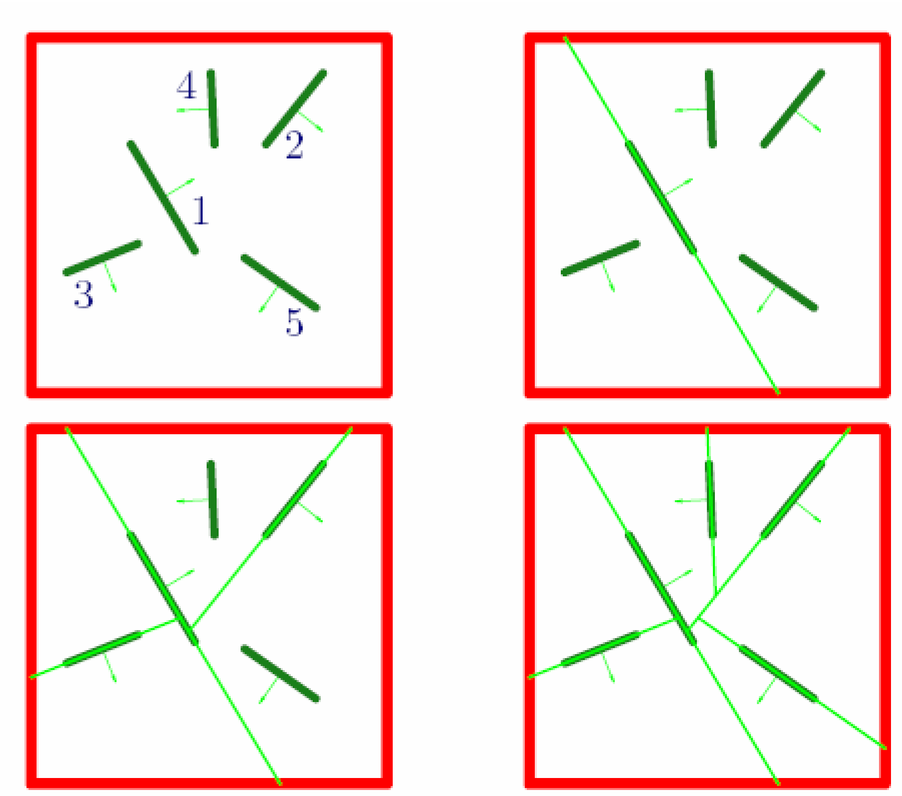
- In closed polyhedron you don't see object "back" faces

- Assumption
  - Normals of faces point *out* from the object



University of British Columbia

BSP Trees



- Convention: Right sibling in N_p direction
- BSP Tree is ***view independent***
- Constructed using only object geometry
- Can be used in hidden surface removal from multiple views
- How to choose what is visible for given view?

University of British Columbia

Z-Buffer

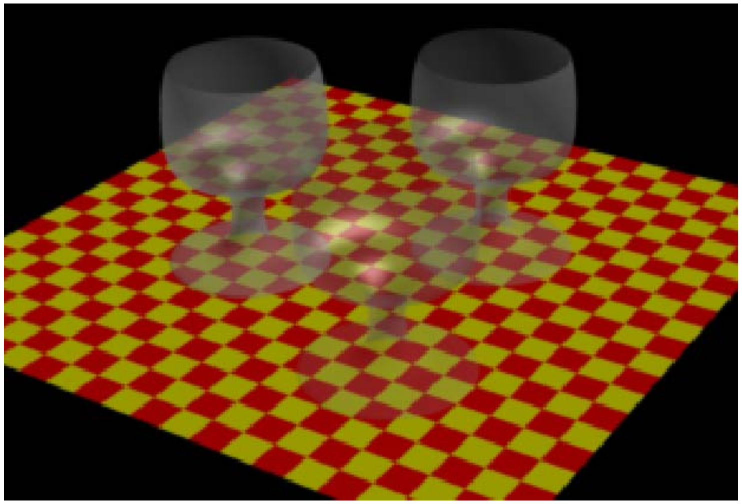
```
ZBuffer(Scene)
For every pixel (x,y) do PutZ(x,y,MaxZ);
For each polygon P in Scene do
   Q := Project(P);
   For each pixel (x,y) in Q do
      z1 := Depth(Q,x,y);
      if (z1<GetZ(x,y)) then
         PutZ(x,y,z1);
         PutColor(x,y,Col(P));
      end;
   end;
end;
```

- Questions: How to compute Project(P) & Depth(Q,x,y)?

University of British Columbia

Transparency/Object Buffer

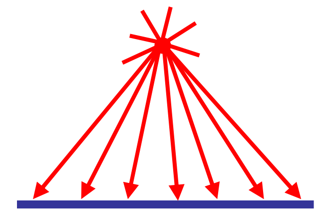
- A-buffer - extension to Z-buffer
- Save all pixel values
- At the end – have list of polygons & depths (order) for each pixel
- Simulate transparency by weighting different list elements



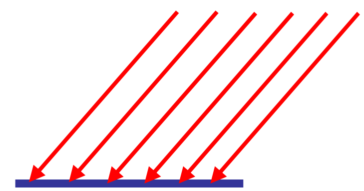
University of British Columbia

Light Sources

- Point source
  - light originates at a point
  - Rays hit planar surface at different angles



- Parallel source
  - light rays are parallel
  - Rays hit a planar surface at identical angles
  - May be modeled as point source at infinity
  - *Directional light*



University of
British Columbia

Light

- Light has color
- Interacts with object color (r,g,b)

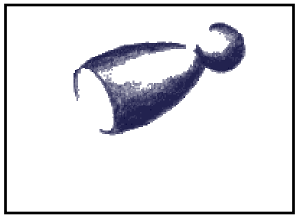
$$I = I_a k_a$$

$$I_a = (I_{ar}, I_{ag}, I_{ab})$$

$$k_a = (k_{ar}, k_{ag}, k_{ab})$$

$$I = (I_r, I_g, I_b) = (I_{ar}k_{ar}, I_{ag}k_{ag}, I_{ab}k_{ab})$$

- Blue light on white surface?
- Blue light on red surface?



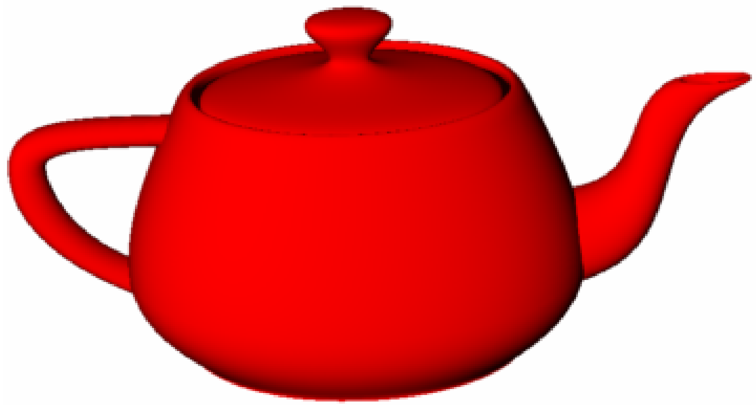
University of
British Columbia

Diffuse Reflection

- Illumination equation is now:

$$I = I_a k_a + I_p k_d (N \cdot L) = I_a k_a + I_p k_d \cos\theta$$

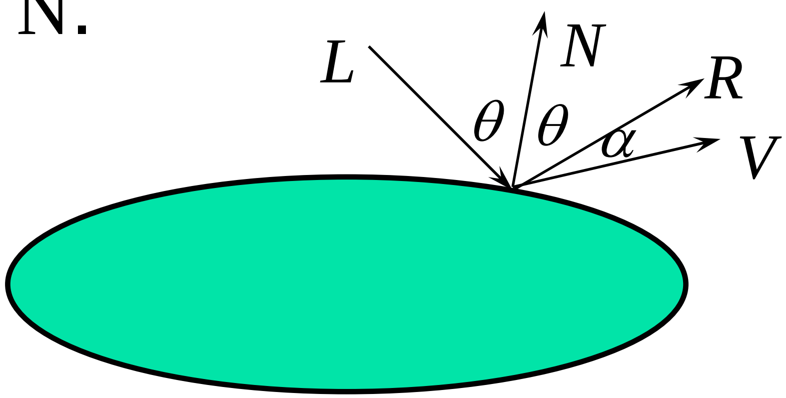
- I_p - point/parallel source's intensity
- k_d - surface diffuse reflection coefficient



- Can we locate light source from shading?

Specular Reflection

- Shiny objects (e.g. metallic) reflect light in preferred direction R determined by surface normal N.



- Most objects are not ideal mirrors - reflect in the immediate vicinity of R

University of
British Columbia

Illumination Equation

- For multiple light sources:

$$I = I_a k_a + \sum_p \frac{I_p}{d_p{}^2}\left(k_d (N \cdot L_p) + k_s (R_p \cdot V)^n\right)$$

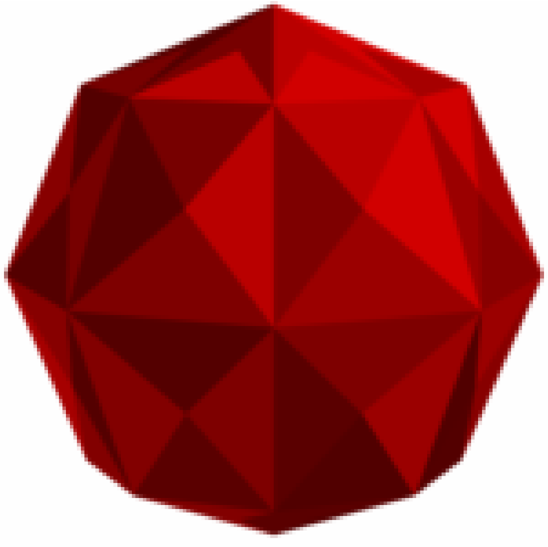
- d_p - distance between surface and light source + distance between surface and viewer (Heuristic atmospheric attenuation)

University of British Columbia

shadingmodel

Flat Shading

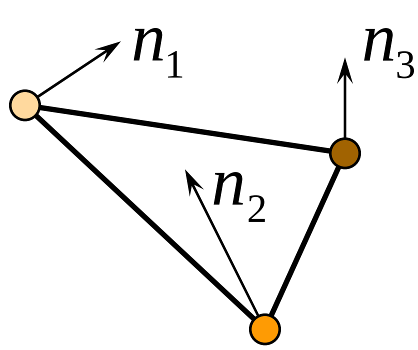
- Illumination value depends only on polygon normal
  - each polygon colored with uniform intensity
- Not adequate for polygons approximating smooth surface
- Looks non-smooth
  - worsened by Mach bands effect

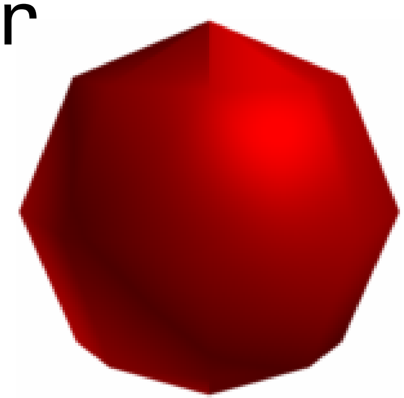


University of British Columbia

Gourard Shading

- Polyhedron - approximation of smooth surface
  - Assign to each vertex normal of original surface at point
  - If surface not available use estimate normal
- Compute illumination intensity at vertices using those normals
- Linearly interpolate vertex intensities over interior pixels of polygon projection

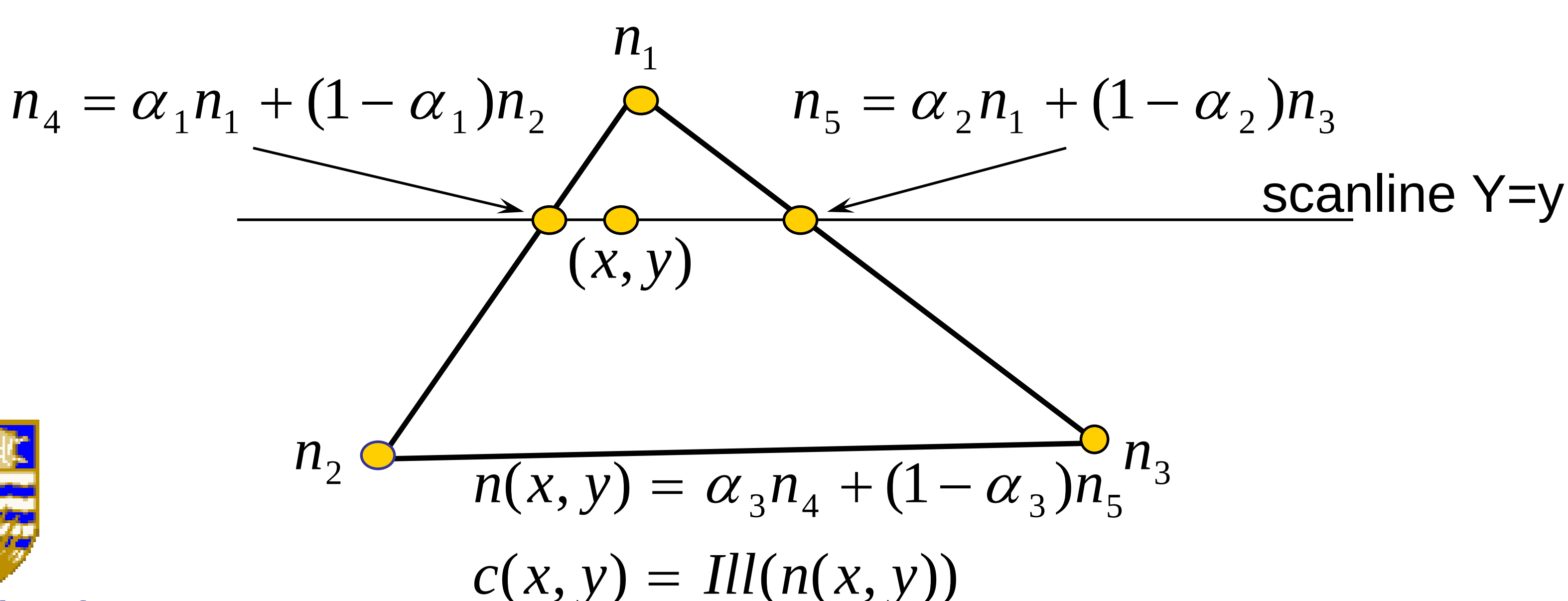




University of British Columbia

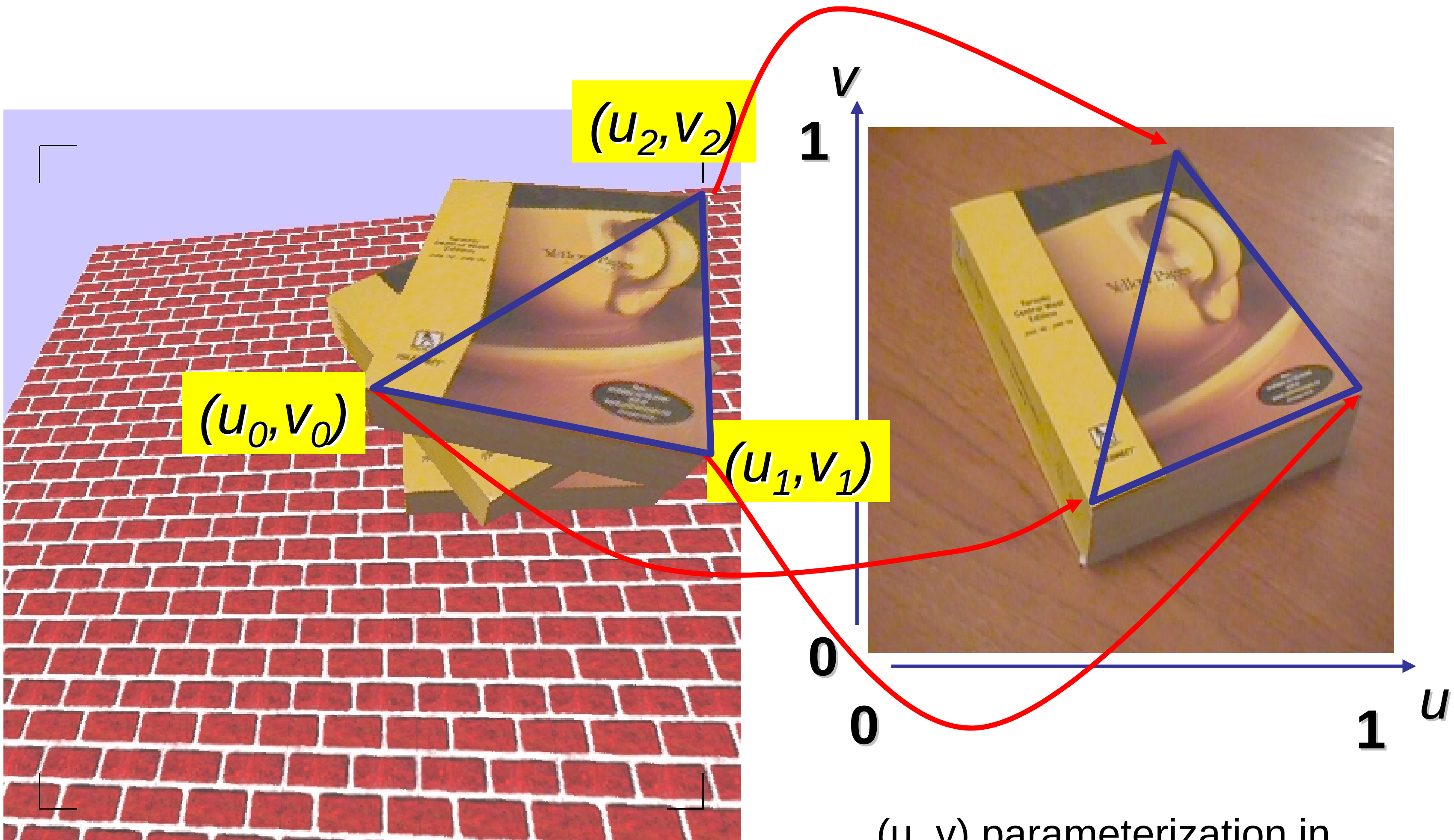
Phong Shading

- Interpolate (in image space) normal vectors instead of intensities
- Apply illumination equation for each interior pixel with its own normal



University of British Columbia

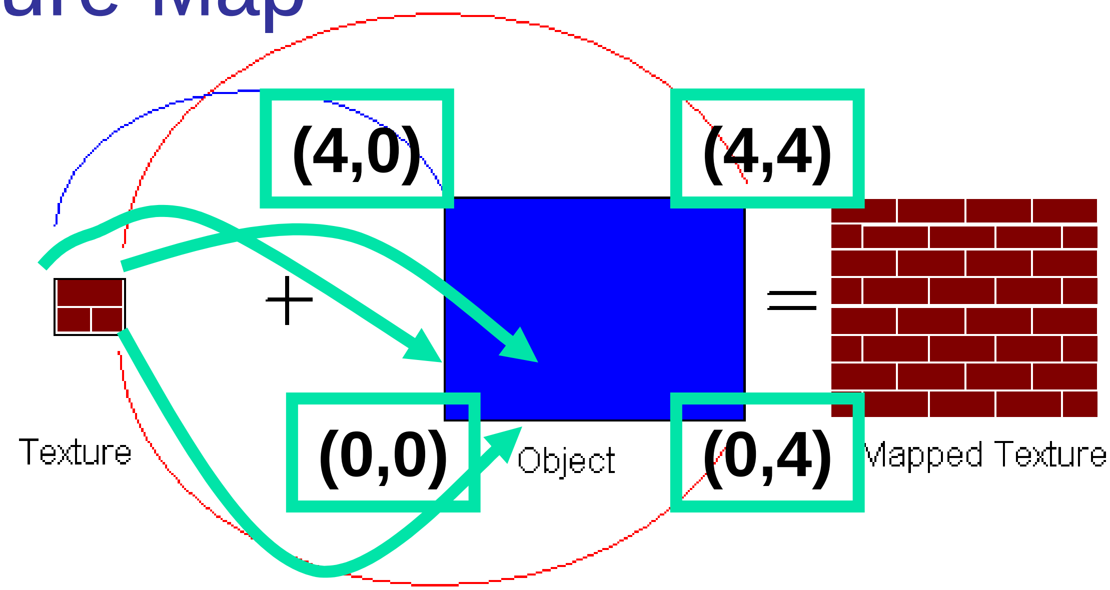
Texture Mapping



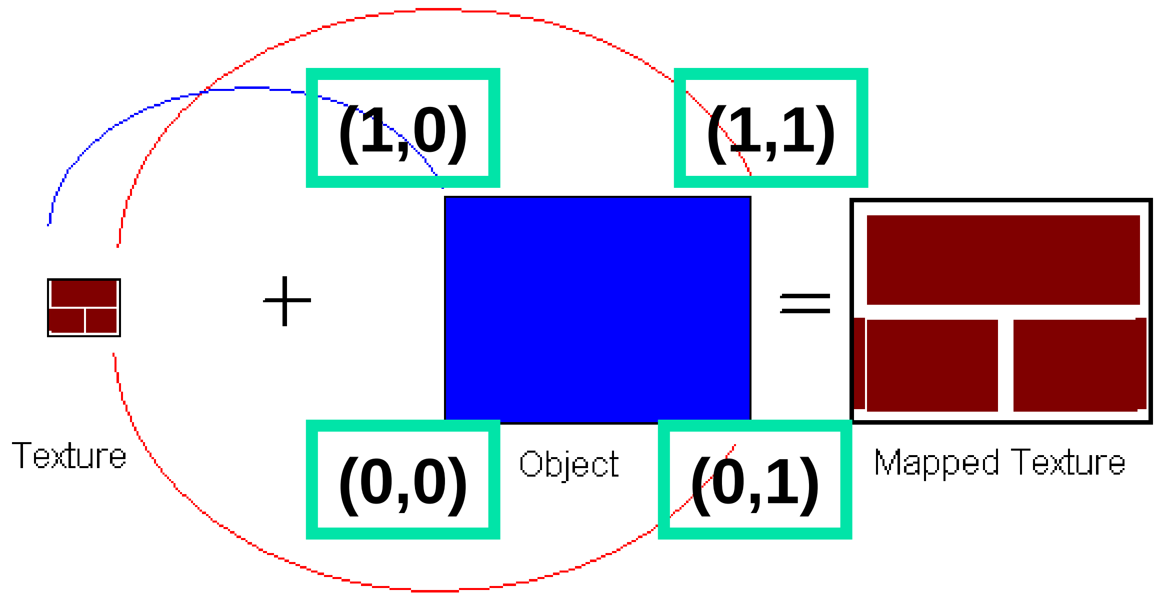
(u, v) parameterization in OpenGL

Example Texture Map

glTexCoord2d(4, 4);
glVertex3d (x, y, z);



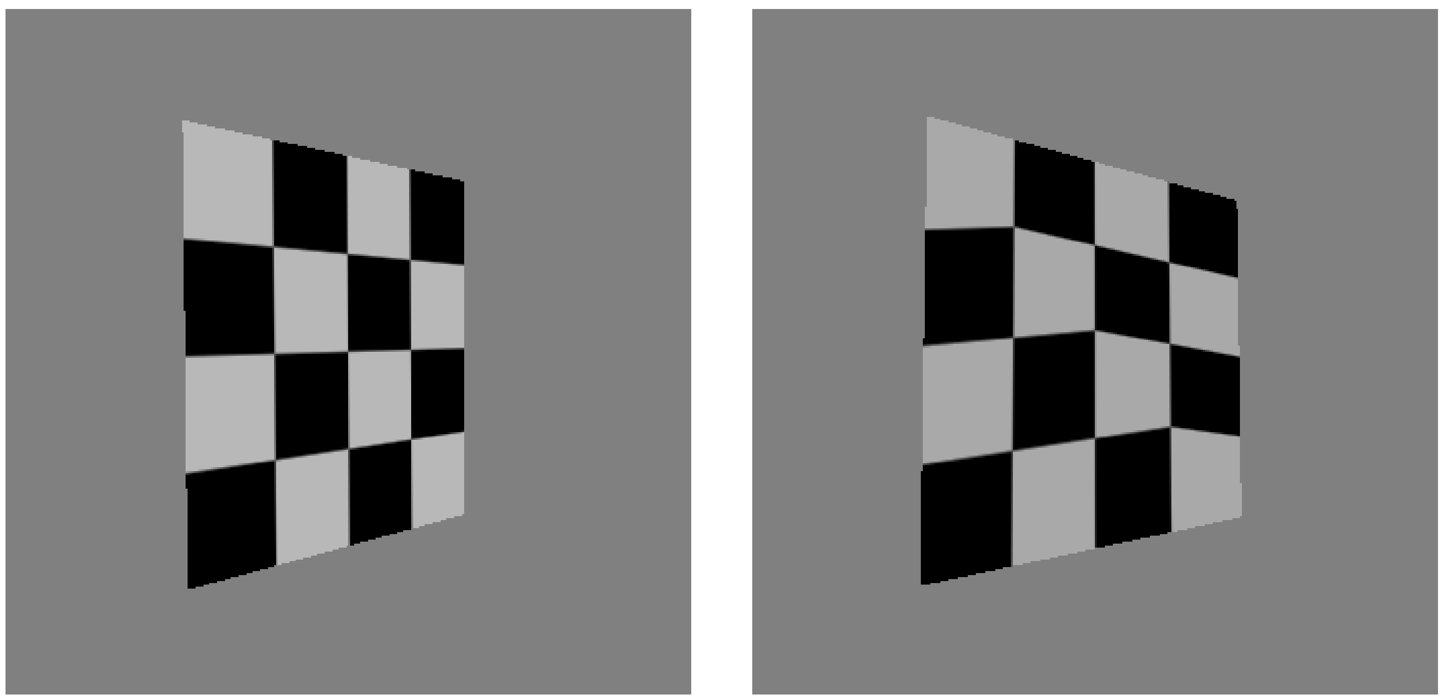
glTexCoord2d(1, 1);
glVertex3d (x, y, z);



University of
British Columbia

Texture Mapping

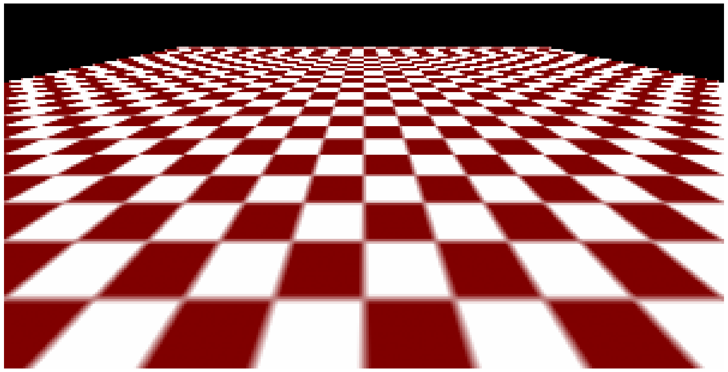
- Texture coordinate interpolation
  - Perspective foreshortening problem
  - Also problematic for color interpolation, etc.



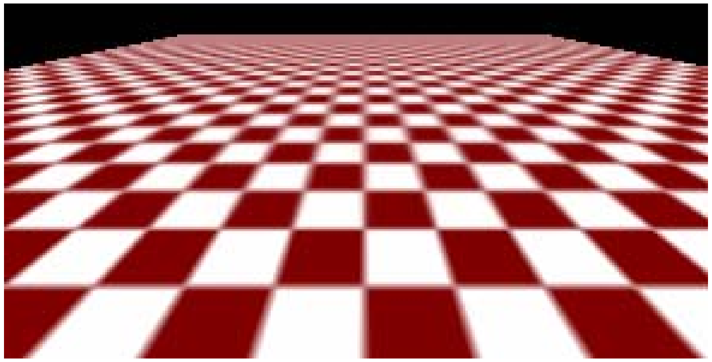
University of
British Columbia

MIP-mapping

without



with



University of
British Columbia

Volumetric Texture - Principles

- 3D function ρ
  - $\rho = \rho(x,y,z)$
- Texture Space – 3D space that holds the texture (discrete or continuous)
- Rendering: for each rendered point P(x,y,z) compute $\rho(x,y,z)$
- Volumetric texture mapping function/space transformed with objects

Texture Parameters

- In addition to color can control other material/object properties
  - Reflectance (either diffuse or specular)
  - Surface normal (bump mapping)
  - Transparency
  - Reflected color (environment mapping)

University of
British Columbia

Environment Mapping: Cube Mapping

- 6 planar textures, sides of cube
  - point camera in 6 different directions, facing out from origin