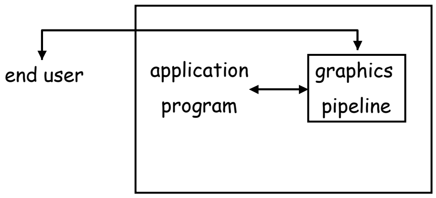
cs155 – z sweedyk

graphics
pipeline
systems

who's who



end user

application program

graphics pipeline

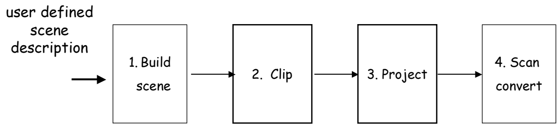
who's who for today

user ⟷ graphics pipeline

user defined scene description

- models
- lights
- view (eye/camera)

graphics pipeline



user defined scene description

1. Build scene

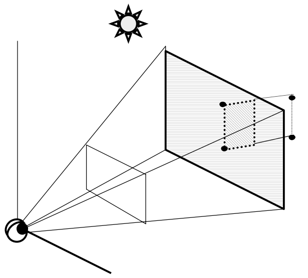
2. Clip

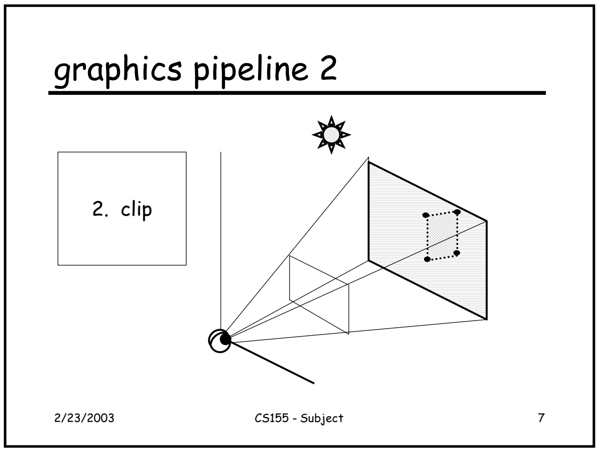
3. Project

4. Scan convert

graphics pipeline 1

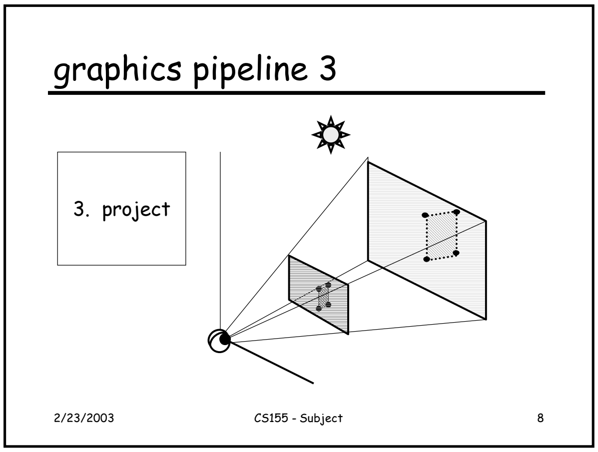
1. build scene





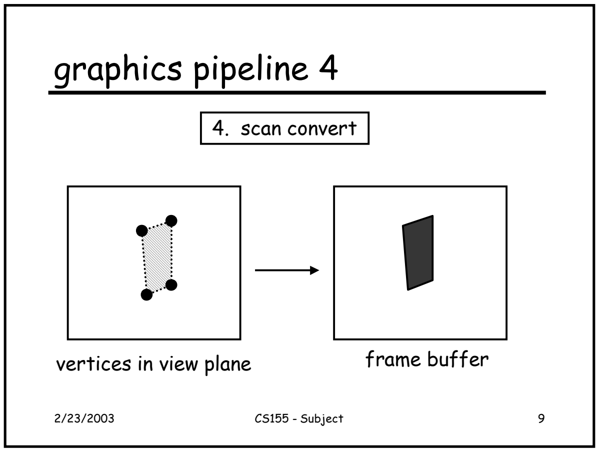
graphics pipeline 2

2. clip



graphics pipeline 3

3. project

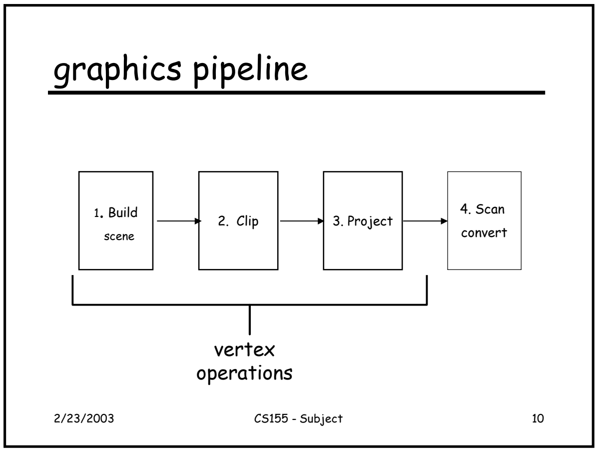


graphics pipeline 4

4. scan convert

vertices in view plane

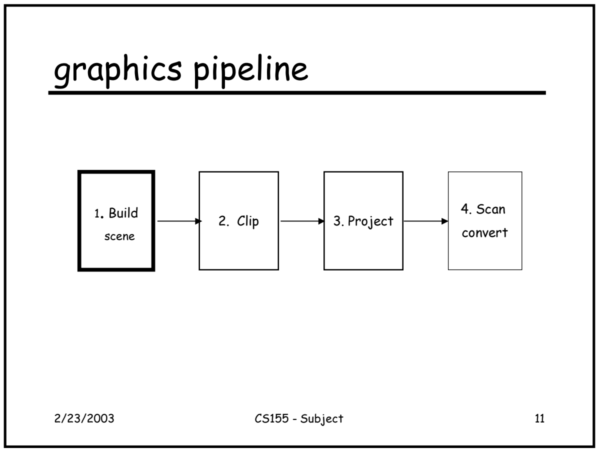
frame buffer



graphics pipeline

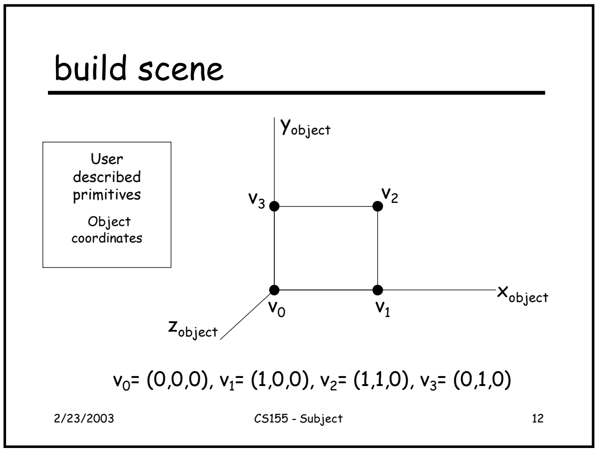
1. Build scene → 2. Clip → 3. Project → 4. Scan convert

vertex operations



graphics pipeline

1. Build scene → 2. Clip → 3. Project → 4. Scan convert



build scene

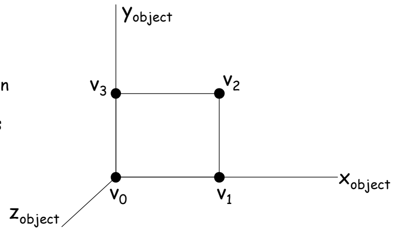
User described primitives

Object coordinates

v_0= (0,0,0), v_1= (1,0,0), v_2= (1,1,0), v_3= (0,1,0)

build scene

pipeline representation is in homogeneous coordinates



v_0= (0,0,0,1), v_1= (1,0,0,1), v_2= (1,1,0,1), v_3= (0,1,0,1)

primitives

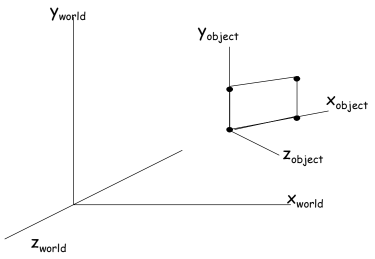
- points
- line segments
- polygons

build scene

User described modeling transforms

World coordinates

Vertex in world coordinates: $M_W v$



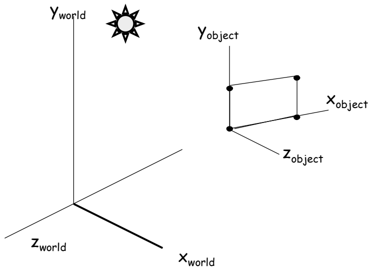
modeling transforms

- scale
- rotate
- translate

build scene

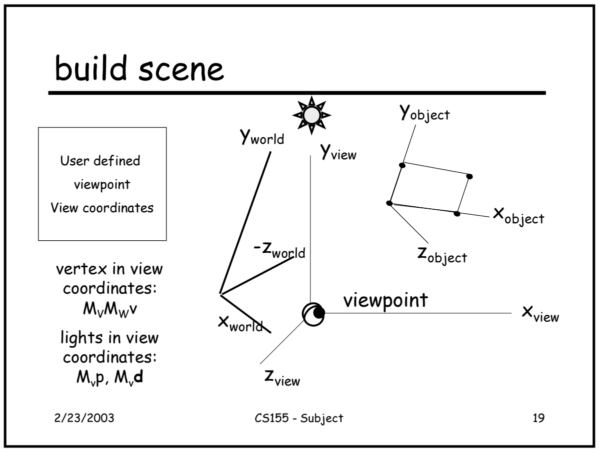
User described lights

World coordinates



Lights

- Ambient
- Directional
- Point
- Spot



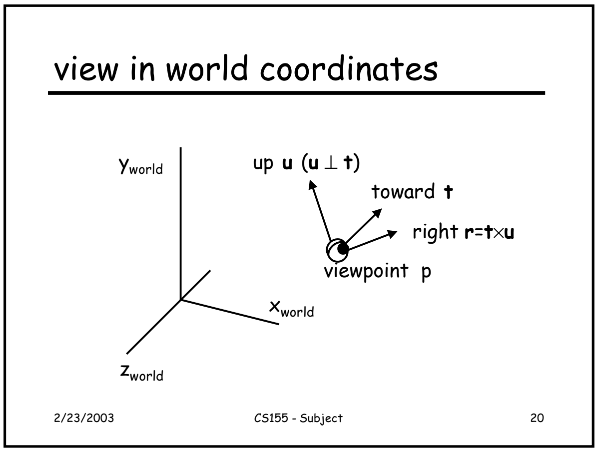
build scene

User defined
viewpoint
View coordinates

vertex in view coordinates: $M_V M_W v$

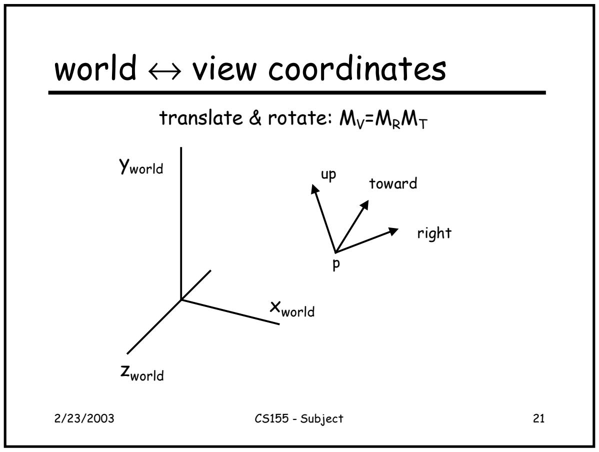
lights in view coordinates: $M_v p$, $M_v \mathbf{d}$

2/23/2003 CS155 - Subject 19



view in world coordinates

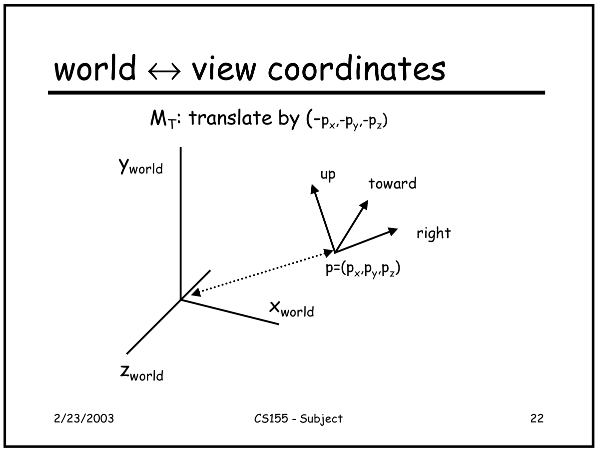
2/23/2003 CS155 - Subject 20



world ↔ view coordinates

translate & rotate: $M_V=M_R M_T$

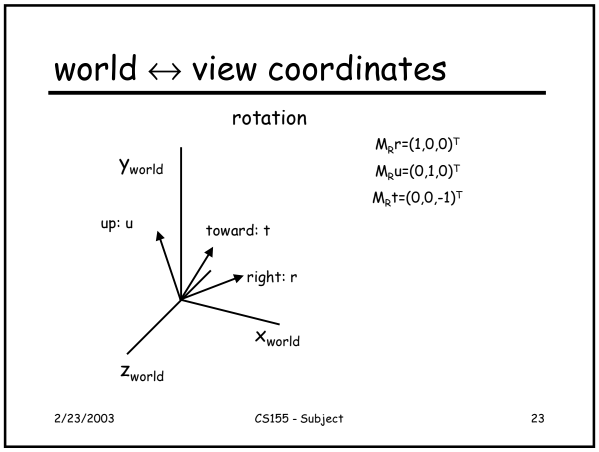
2/23/2003 CS155 - Subject 21



world ↔ view coordinates

M_T: translate by $(-p_x,-p_y,-p_z)$

2/23/2003 CS155 - Subject 22



world ↔ view coordinates

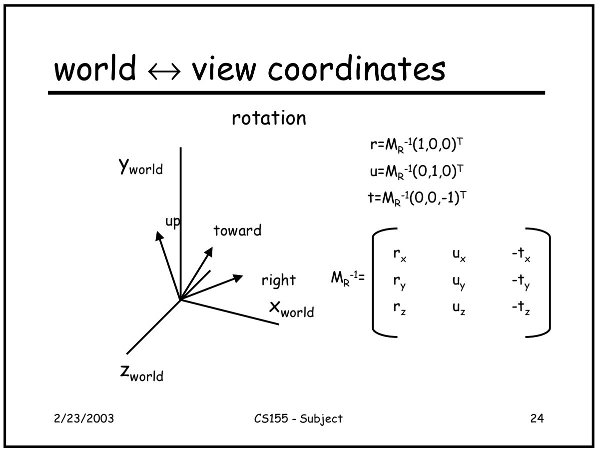
rotation

$M_R r=(1,0,0)^T$

$M_R u=(0,1,0)^T$

$M_R t=(0,0,-1)^T$

2/23/2003 CS155 - Subject 23



world ↔ view coordinates

rotation

$r=M_R^{-1}(1,0,0)^T$

$u=M_R^{-1}(0,1,0)^T$

$t=M_R^{-1}(0,0,-1)^T$

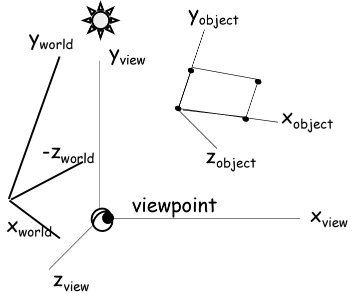
$$M_R^{-1}=\begin{pmatrix} r_x & u_x & -t_x \\ r_y & u_y & -t_y \\ r_z & u_z & -t_z \end{pmatrix}$$

2/23/2003 CS155 - Subject 24

build scene

vertex in view coordinates: $M_V M_W v$

lights in view coordinates: $M_v p$, $M_v \mathbf{d}$



geometric primitives

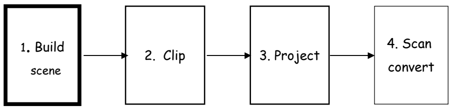
object coordinates: v	description of vertex
world coordinates: $M_W v$	description of vertex situated in world
view coordinates: $M_V M_W v$	description of vertex in world as seen from a particular viewpoint

lights

world coordinates: p, $\mathbf{d}$	description of light position/direction in world
view coordinates: $M_V p$, $M_v \mathbf{d}$	description of light position/direction in world as seen from a particular viewpoint

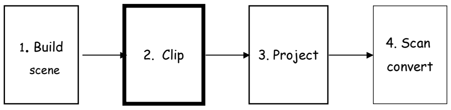
note: $M_v \mathbf{d}$ is shorthand for the "multiply vector" operation we've used before!

graphics pipeline



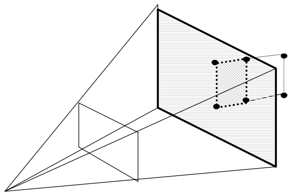
Done

graphics pipeline

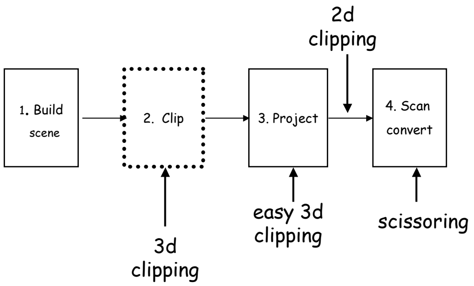


graphics pipeline 2

user specified view volume



when to clip?



2d clipping

1. Build scene
2. Clip
3. Project
4. Scan convert

3d clipping

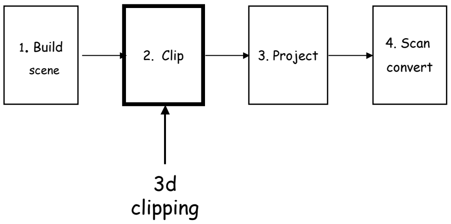
easy 3d clipping

scissoring

---

when to clip?

this is most general … we'll say more about the other approaches later



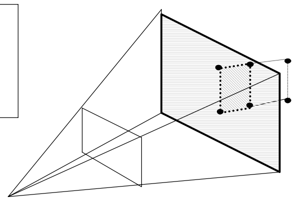
1. Build scene
2. Clip
3. Project
4. Scan convert

3d clipping

---

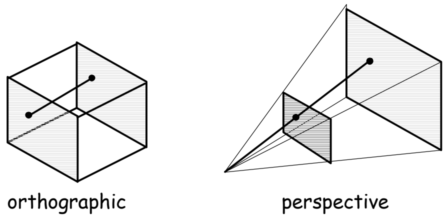
graphics pipeline 2

user specified view volume



---

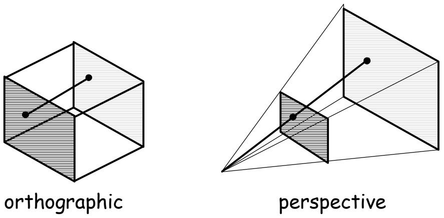
projection type & view volume



orthographic perspective

---

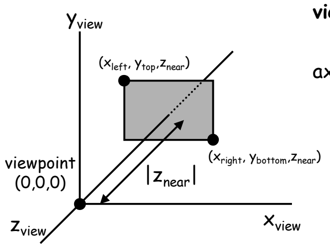
view window



orthographic perspective

---

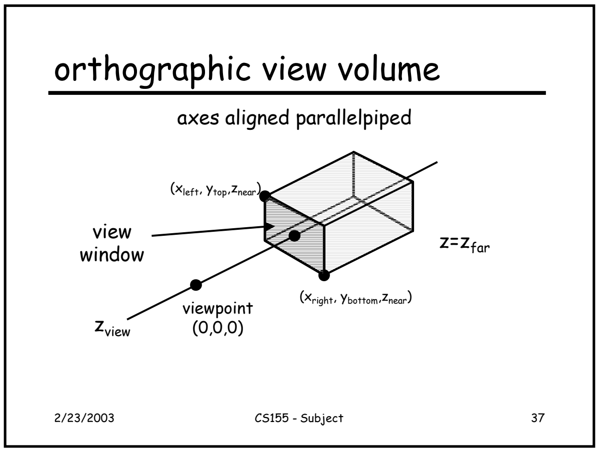
view window specification



view window (in view coordinates):

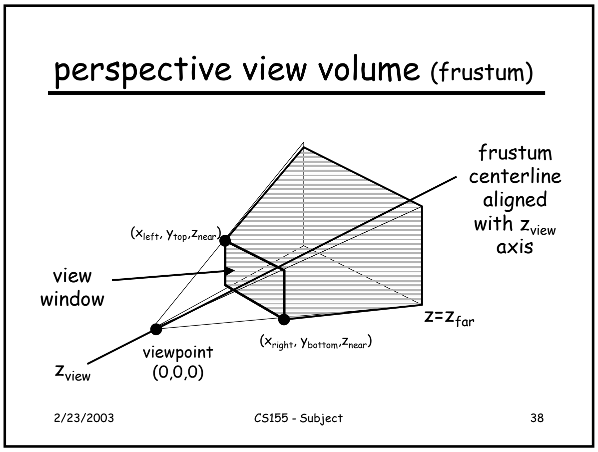
axes aligned rectangle

- in view plane $z=z_{near}$
- centered at z axis
- with boundaries $x=x_{left}$, $x=x_{right}$, $y=y_{bottom}$, $y=y_{top}$

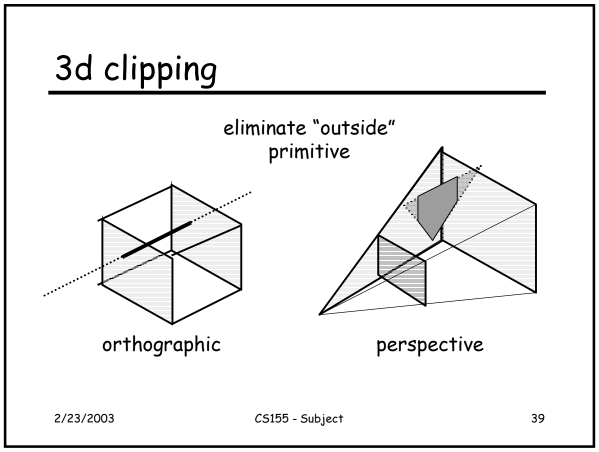


orthographic view volume

axes aligned parallelpiped

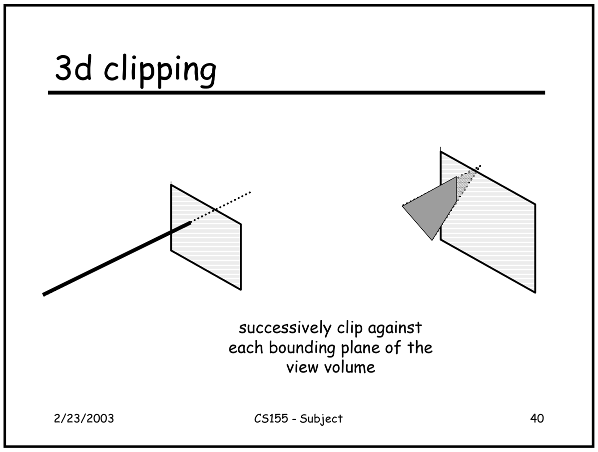


perspective view volume (frustum)



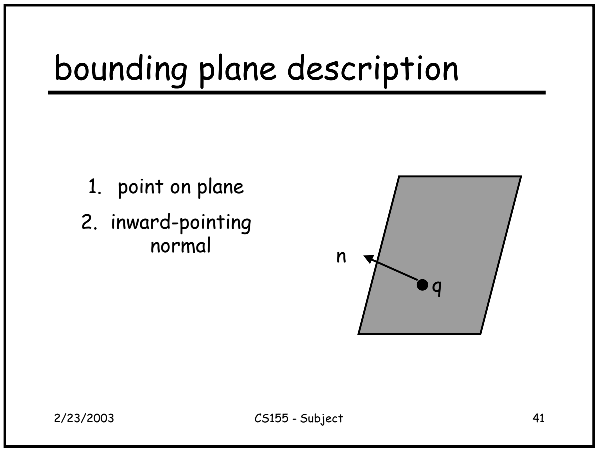
3d clipping

eliminate “outside” primitive



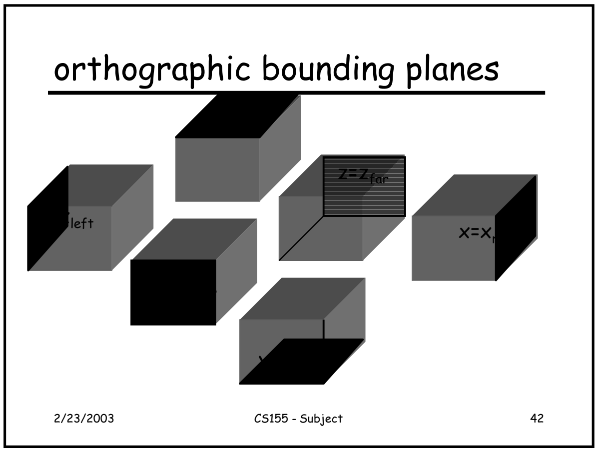
3d clipping

successively clip against each bounding plane of the view volume

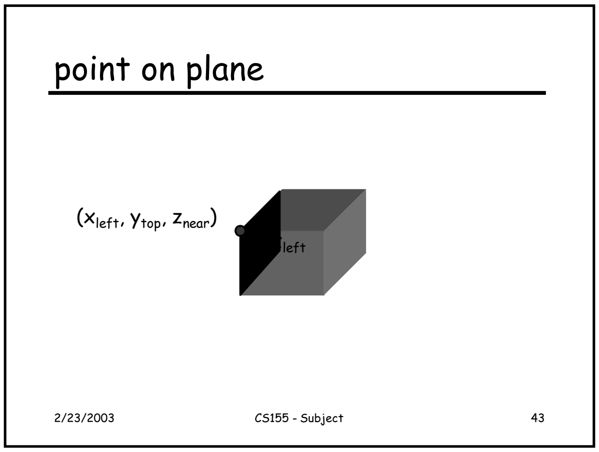


bounding plane description

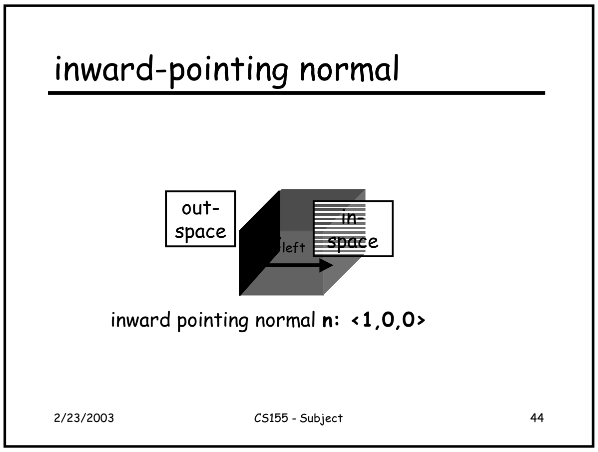
1. point on plane
2. inward-pointing normal



orthographic bounding planes

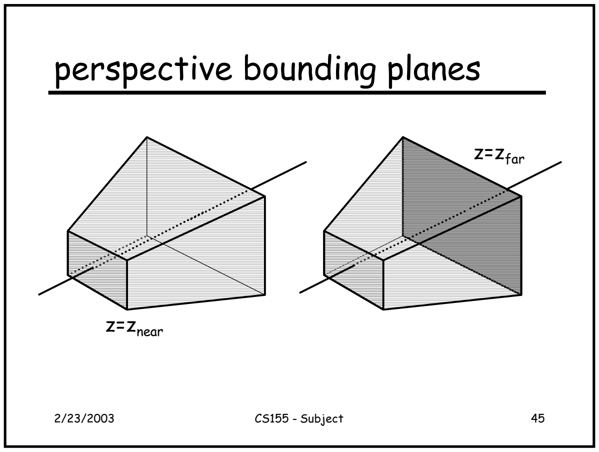


point on plane

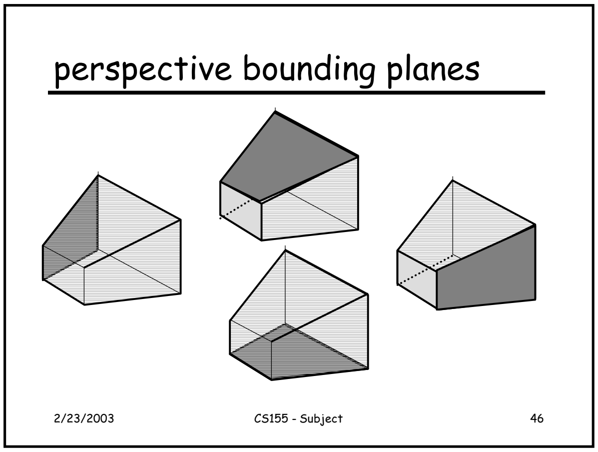


inward-pointing normal

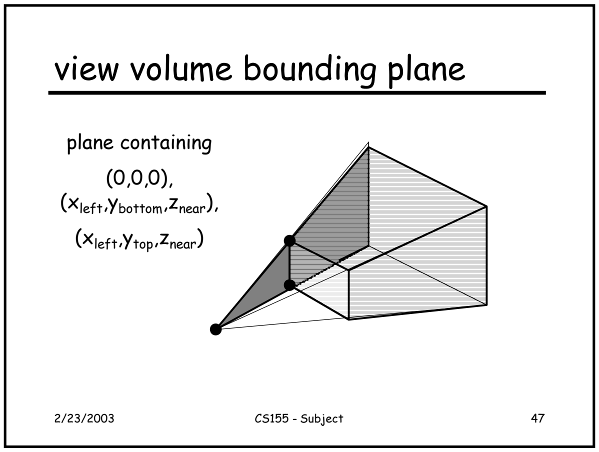
inward pointing normal **n: <1,0,0>**



perspective bounding planes

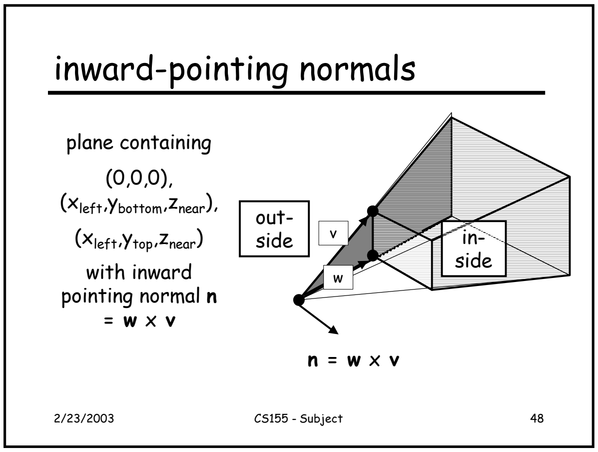


perspective bounding planes



view volume bounding plane

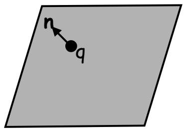
plane containing
(0,0,0),
$(x_{left},y_{bottom},z_{near})$,
$(x_{left},y_{top},z_{near})$



inward-pointing normals

plane containing
(0,0,0),
$(x_{left},y_{bottom},z_{near})$,
$(x_{left},y_{top},z_{near})$
with inward
pointing normal **n**
= **w** × **v**

clipping plane



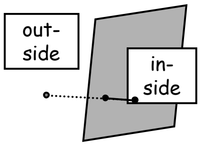
clipping plane specification:

- point q on the plane
- inward pointing normal **n**

3d clipping

given a clipping plane and a graphics primitive

return "in-side primitive"



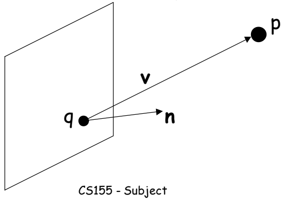
3d clipping

- vertex clipping
- line segment clipping
- polygon clipping

vertex clipping

p is in with respect to the clipping plane iff $\mathbf{n} \bullet \mathbf{v} \geq 0$ where

- **n** is the inward facing normal
- **v** is the vector from q to p

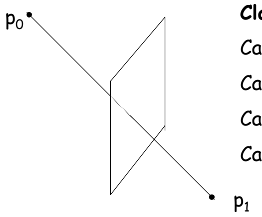


3d clipping

- vertex clipping
- **line segment clipping**
- polygon clipping

line segment clipping

use test for vertex clipping



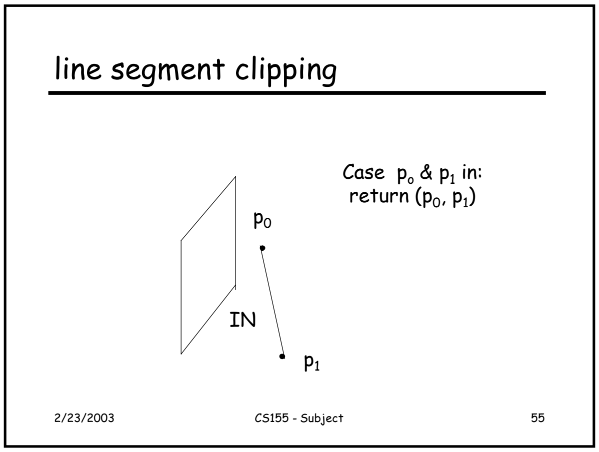
Classify endpoint p_o & p_1

Case: p_o & p_1 in ______

Case: p_o & p_1 out______

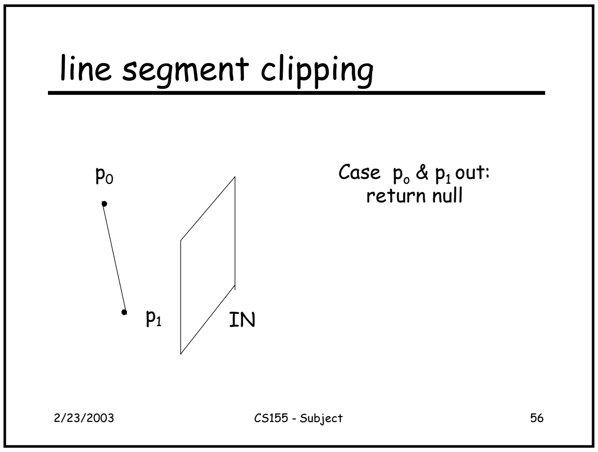
Case: p_o in & p_1 out ______

Case: p_o out & p_1 in ______



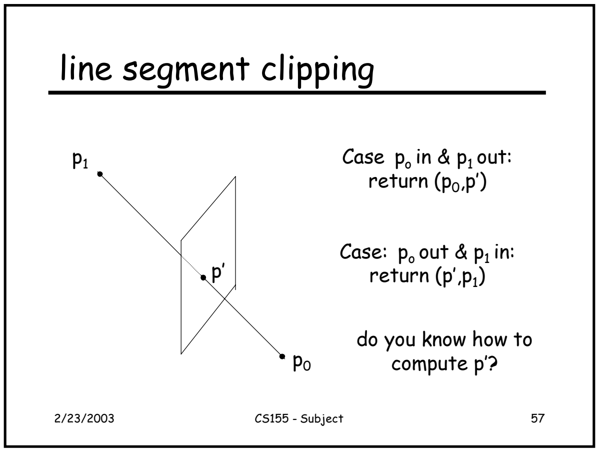
line segment clipping

Case p_o & p_1 in:
return (p_0, p_1)



line segment clipping

Case p_o & p_1 out:
return null

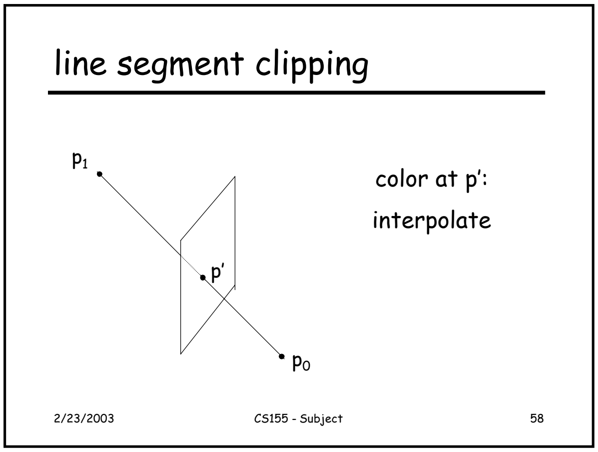


line segment clipping

Case p_o in & p_1 out:
return (p_0,p')

Case: p_o out & p_1 in:
return (p',p_1)

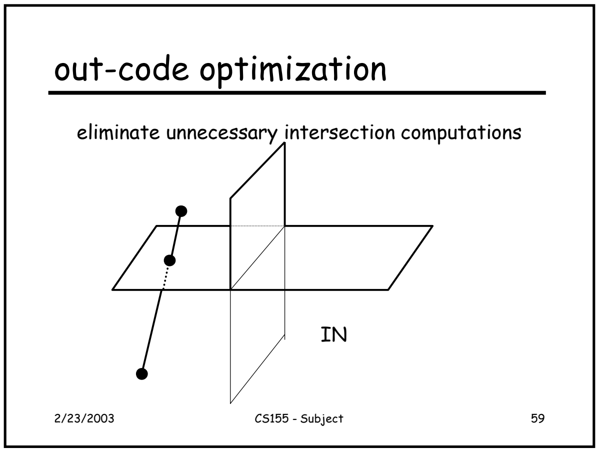
do you know how to
compute p'?



line segment clipping

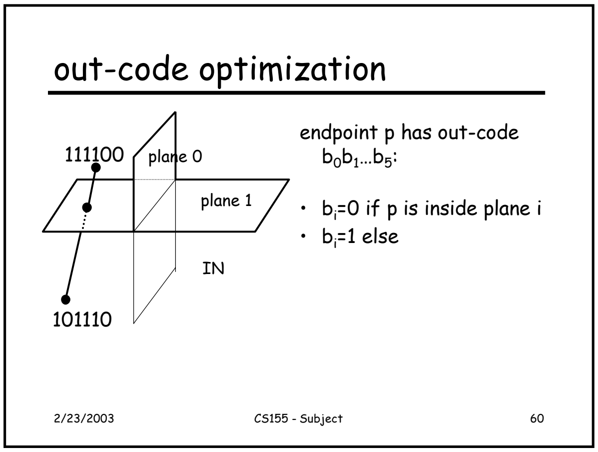
color at p':

interpolate



out-code optimization

eliminate unnecessary intersection computations



out-code optimization

endpoint p has out-code
$b_0b_1 \ldots b_5$:

- b_i=0 if p is inside plane i
- b_i=1 else

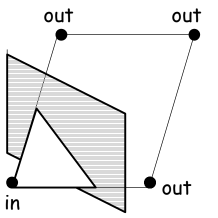
out-code optimization

- compute endpoint out-codes B and B′
- if B AND B′ ≠ 0 return ________
- if B OR B′ = 1 return ________
- else clip against plane j where the i^{th} ___________________
- restart test

3d clipping

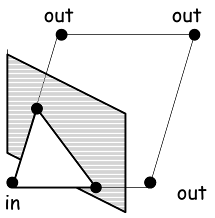
- vertex clipping
- line clipping
- **polygon clipping**

polygon clipping



1. classify vertices

polygon clipping



1. classify vertices
2. compute intersection points of intersecting edges

   &

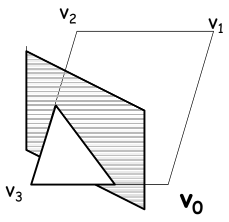
   write out new polygon

polygon clipping

- if v_0 is in then write v_0
- for i=1…n-1
  - case v_i & v_{i+1} in: write v_{i+1}
  - case v_i & v_{i+1} out: do nothing
  - case v_i in and v_{i+1} out: write intersection point
  - case v_i out and v_{i+1} in: write intersection point and v_{i+1}

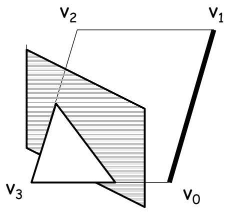
indices taken modulo n

example



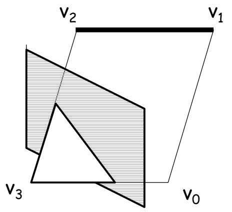
v_0 out: do nothing

example



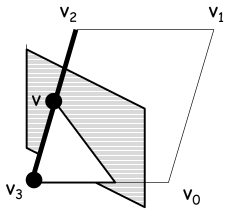
v_0 & v_1 out: do nothing

example



v_1 & v_2 out: do nothing

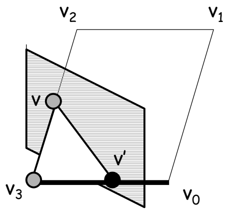
example



v_2 out & v_3 in:

write v, v_3

example



v_3 in & v_0 out:

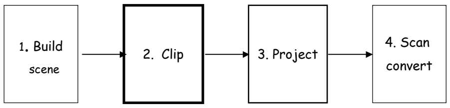
write v'

polygon clipping

- if v_0 is in then write v_0
- for i=1…n-1
  - case v_i & v_{i+1} in: write v_{i+1}
  - case v_i & v_{i+1} out: do nothing
  - case v_i in and v_{i+1} out: write intersection point
  - case v_i out and v_{i+1} in: write intersection point and v_{i+1}

interpolate along edge to find color at intersection point

graphics pipeline



Done