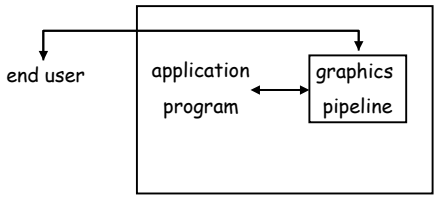
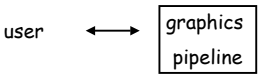
cs155 – z sweedyk

graphics
pipeline
systems

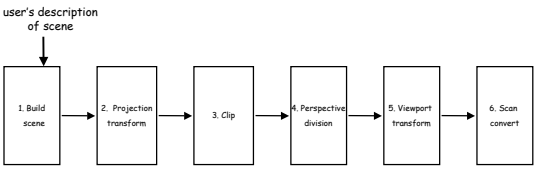
who's who



who's who for today



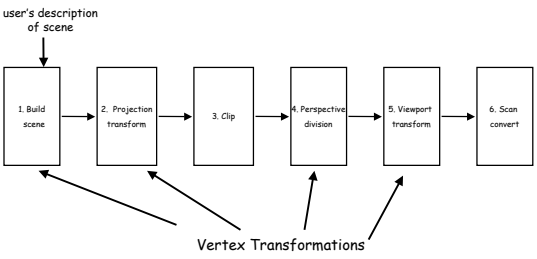
graphics pipeline



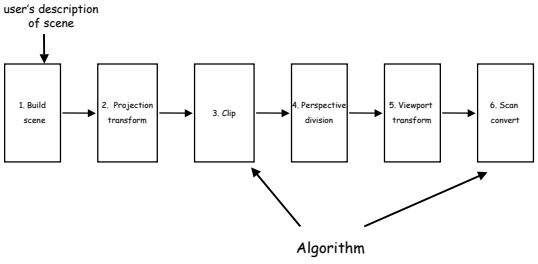
"vertex based" primitives

- points
- lines
- triangles
- convex polygons

graphics pipeline



graphics pipeline



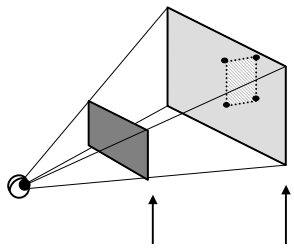
graphics pipeline overview

- simplified pipeline
- general pipeline

graphics pipeline overview

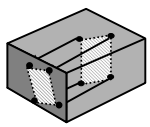
- simplified pipeline
  - geometric primitives defined in standardized, homogenous world coordinates
  - orthographic projection
  - standardized view volume
- general pipeline

perspective projection



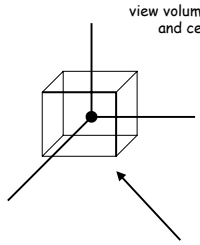
pipeline: view volume bounded by near and far planes

orthographic (parallel) projection



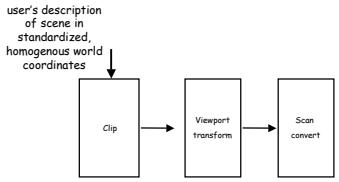
standard orthographic view volume

view volume is 2x2x2, axis-aligned, and centered at the origin

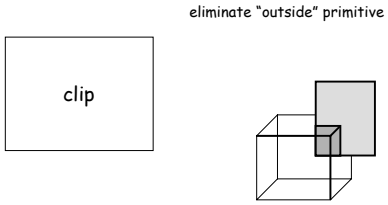


rendered image is the parallel projection of scene onto front plane

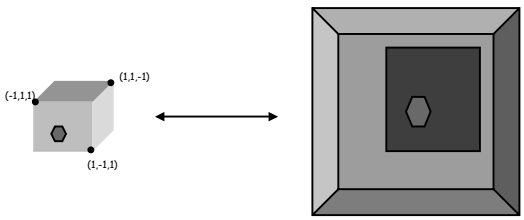
simplified graphics pipeline



graphics pipeline: clip



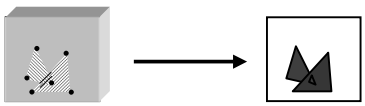
viewport transformation



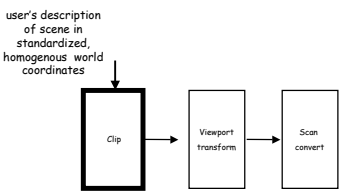
standardized world coordinates

image coordinates

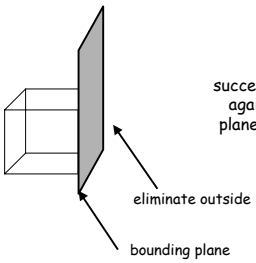
scan conversion



simplified graphics pipeline



3d clipping overview

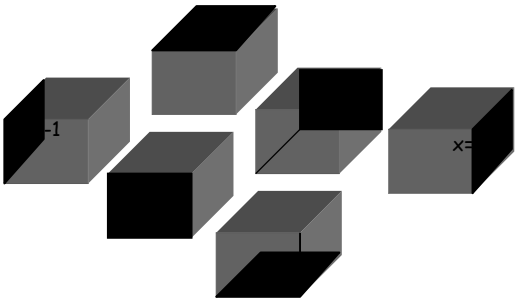


successively clip primitive against each bounding plane of the view volume

eliminate outside

bounding plane

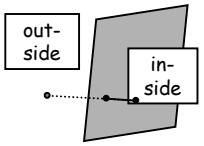
bounding planes of canonical view volume



clipping algorithm

given a clipping plane and a graphics primitive

return "in-side primitive"

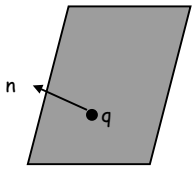


primitives to clip

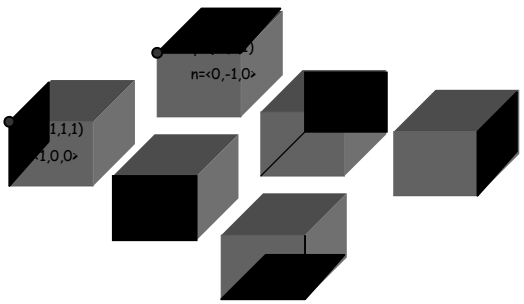
- vertex
- line segment
- polygon

convenient description of bounding plane

1. point on plane q
2. inward-pointing normal n



bounding planes of canonical view volume



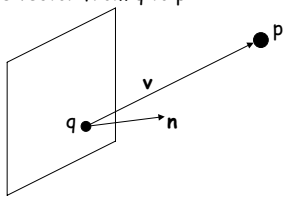
clipping algorithms

- **vertex**
- line segment
- polygon

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

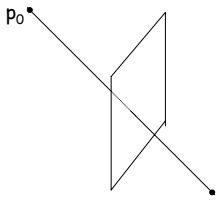


clipping algorithm

- vertex
- **line segment**
- polygon

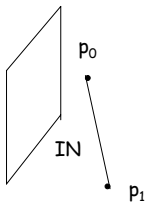
line segment clipping

use test for vertex clipping



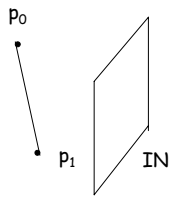
1. **Classify endpoints p_o & p_1 as in or out**

line segment clipping



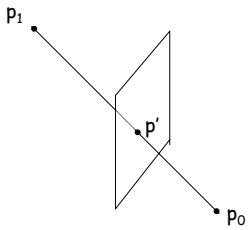
Case p_o & p_1 in:
return (p_0, p_1)

line segment clipping



Case p_o & p_1 out:
return null

line segment clipping



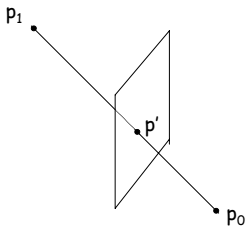
Case p_o in & p_1 out:
return (p_0,p')

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

do you know how to compute p'?

what should happen to w component?

line segment clipping



color at p':

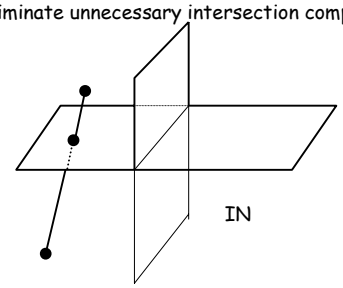
we'll come back to this when we talk about color models

exercise

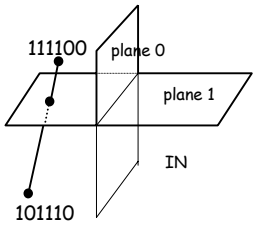
- clip the line with endpoints (-10,-10,-11) and (2,2,1) using the following order of bounding planes:
  - near
  - far
  - left
  - right
  - top
  - bottom
- show the endpoints at the beginning of each step
- how many intersection computations did you do? against which planes?

out-code optimization

eliminate unnecessary intersection computations



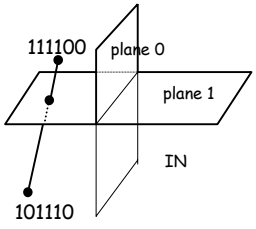
out-code optimization



endpoint p has out-code $b_0b_1...b_5$:

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

out-code optimization

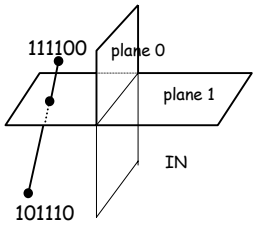


endpoint p has out-code $b_0b_1...b_5$:

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

what does it mean if the i^{th} bit of both endpoints is 1?

out-code optimization



endpoint p has out-code $b_0b_1...b_5$:

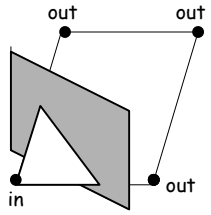
- b_i=0 if p is inside plane i
- b_i=1 else

what does it mean if the i^{th} bit of both endpoints is 0?

clipping algorithm

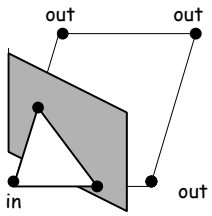
- vertex clipping
- line clipping
- **polygon clipping**

polygon clipping



1. classify vertices

polygon clipping



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

   &

   write out new polygon

polygon clipping

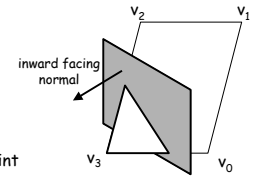
- if v_0 is in then write v_0
- for i=0…n-1
  - case v_i & v_{i+1} in: write v_{i+1}
  - case v_i & v_{i+1} out: do nothing
  - case v_i in and v_{i+1} out: write intersection point
  - case v_i out and v_{i+1} in: write intersection point and v_{i+1}

indices taken modulo n

example

if v_0 is in then write v_0
for i=0…3

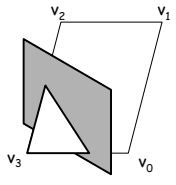
- case v_i & v_{i+1} in:
  write v_{i+1}
- case v_i & v_{i+1} out:
  do nothing
- case v_i in and v_{i+1} out:
  write intersection point
- case v_i out and v_{i+1} in:
  write intersection point and v_{i+1}



example

if v_0 is in then write v_0
for i=0…3

- 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}

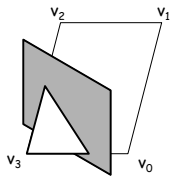


do nothing

example

i=0

- 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}

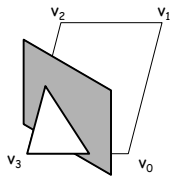


do nothing

example

i=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}

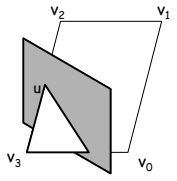


do nothing

example

i=2

- 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}



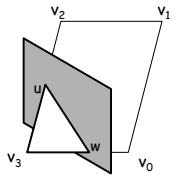
write u, v_3

example

output: u, v_3

i=3

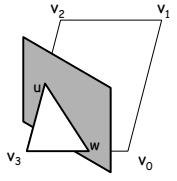
- case v_i & v_{i+1} in:
  write v_{i+1}
- case v_i & v_{i+1} out:
  do nothing
- case v_i in and v_{i+1} out:
  write intersection point
- case v_i out and v_{i+1} in:
  write intersection point
  and v_{i+1}



write w

example

output: u, v_3, w



exercise

- clip the triangle with vertices (-10,0,0), (0,0,0), and (0,10,0) using the following order of bounding planes
  - near (team 1)
  - far (team 2)
  - left (team 3)
  - top (team 1)
  - bottom (team 2)
  - right (team 3)
- each team should write the vertices after each of its steps on the board

simplified graphics pipeline