Computer Graphics

But: un cours de base + 4 TPs “ Do it yourself “

Géraldine Morin

Nombreux slides issus des cours de Scott Schaefer, TAMU

Motivation

- Rendu: partout... applications variées
- GPU (cartes graphiques) programmables

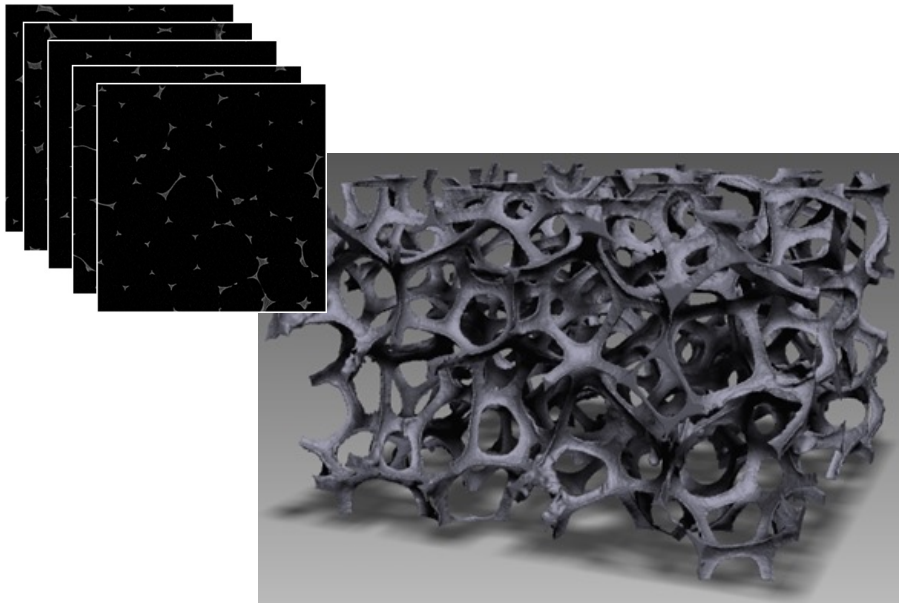
Games

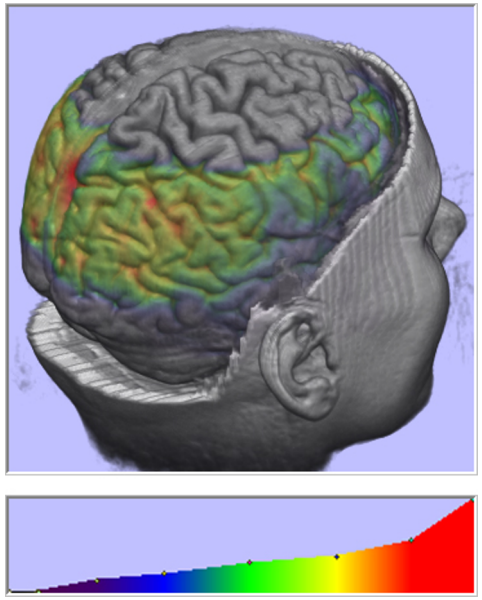

Quests (F9)
Menu (F10)
Allies
Log (F12)
1098
964
24/42
No Upkeep
miss
Transport Ship
1500 / 1500
GAMESPOT

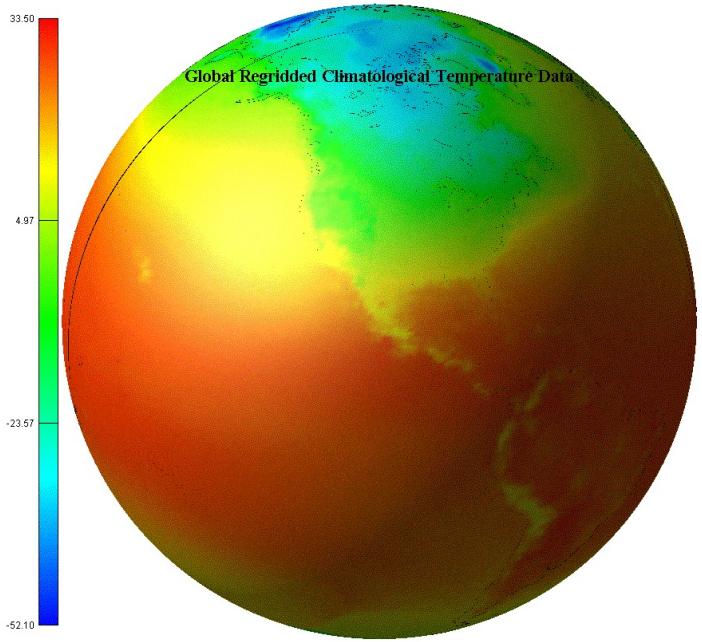

GAMESPO

Movies

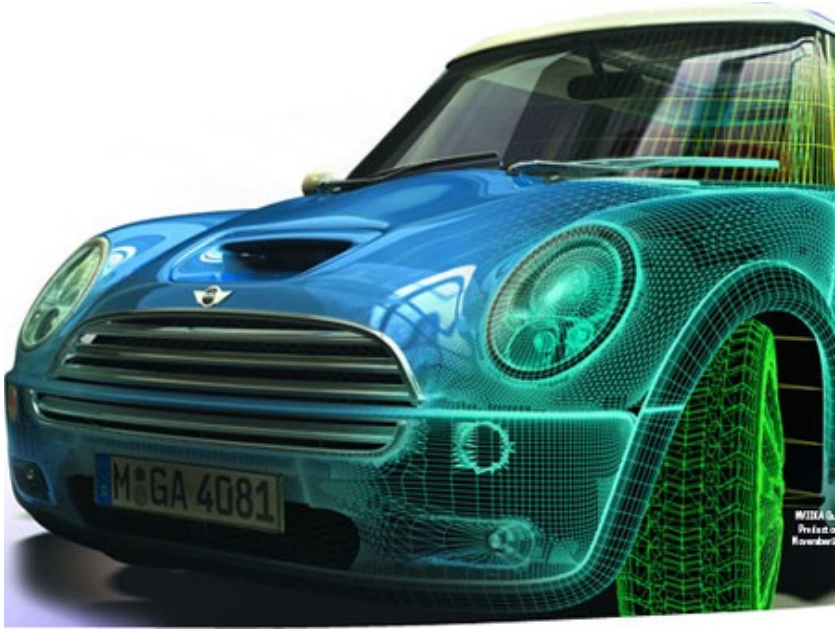
Visualization

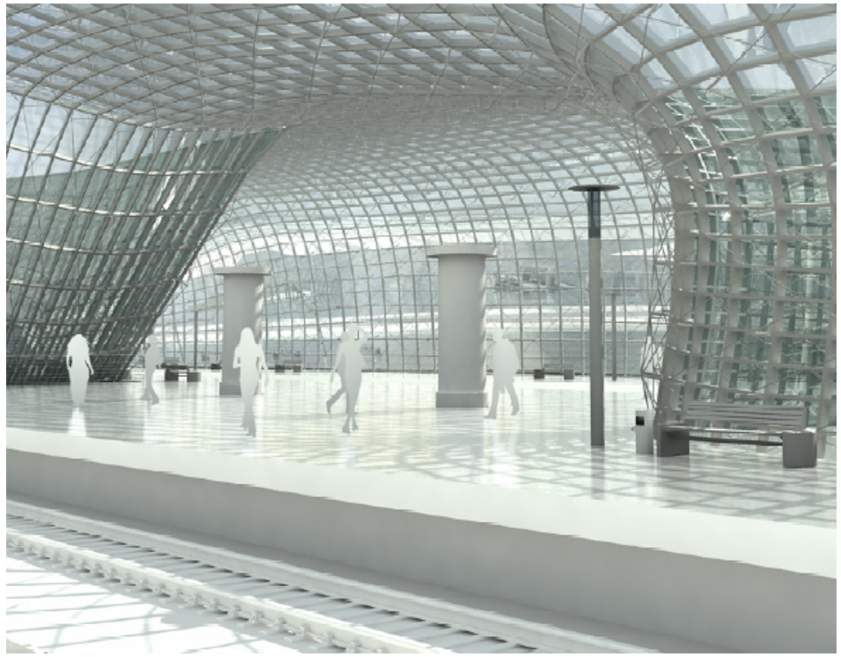






Industrial Design



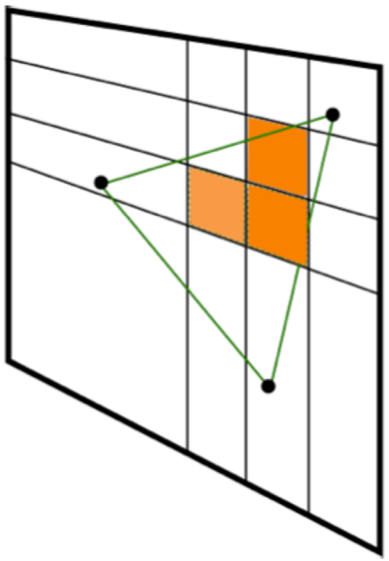


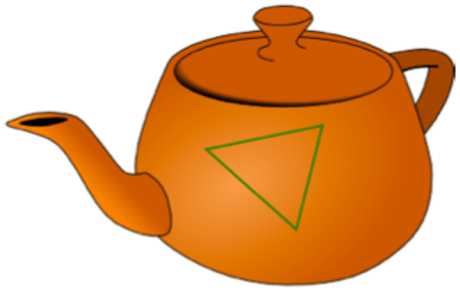
Dans ce cours

- Une introduction
  - 1 cours
    - Le pipeline graphique de base (pas de shaders)
- Une expérience pratique
  - 4 Tps: implémenter un rendu 'basique' à la main

Pipeline graphique

- **Rastérisation**

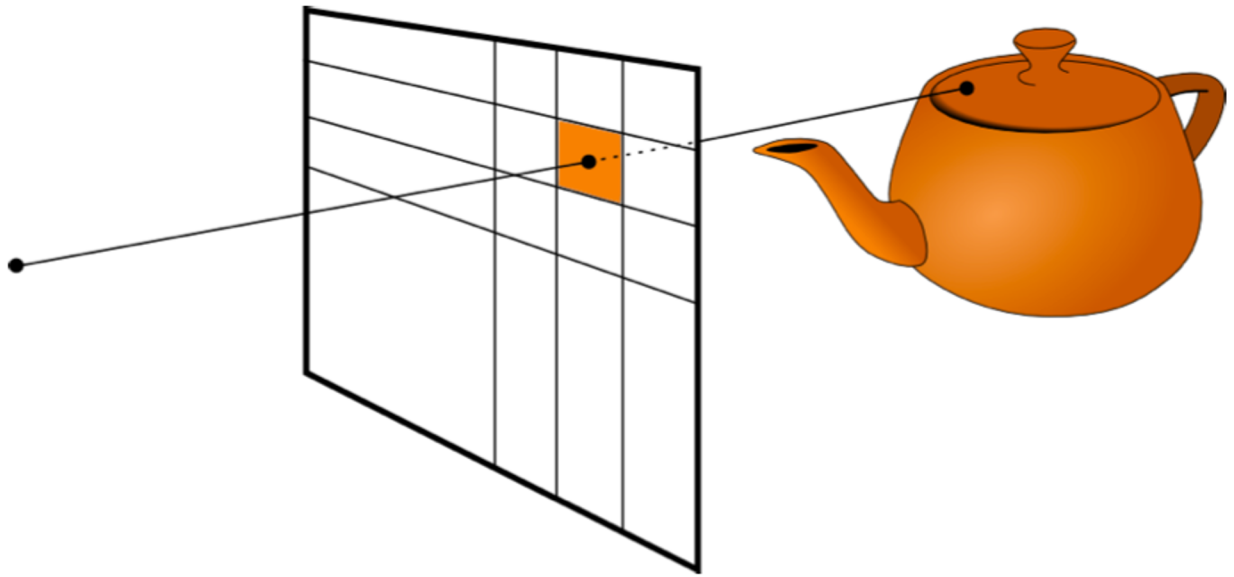




  - « Forward projection »
  - Procédure centrale: remplissage de primitives

Pipeline graphique

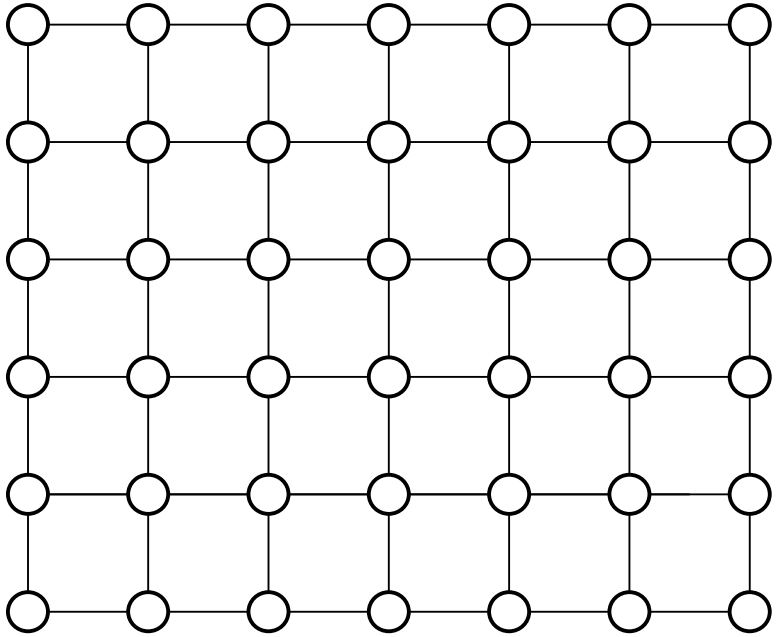
- Lancer de rayons



  - « Backward projection »
  - Procédure centrale: intersection rayon / primitive

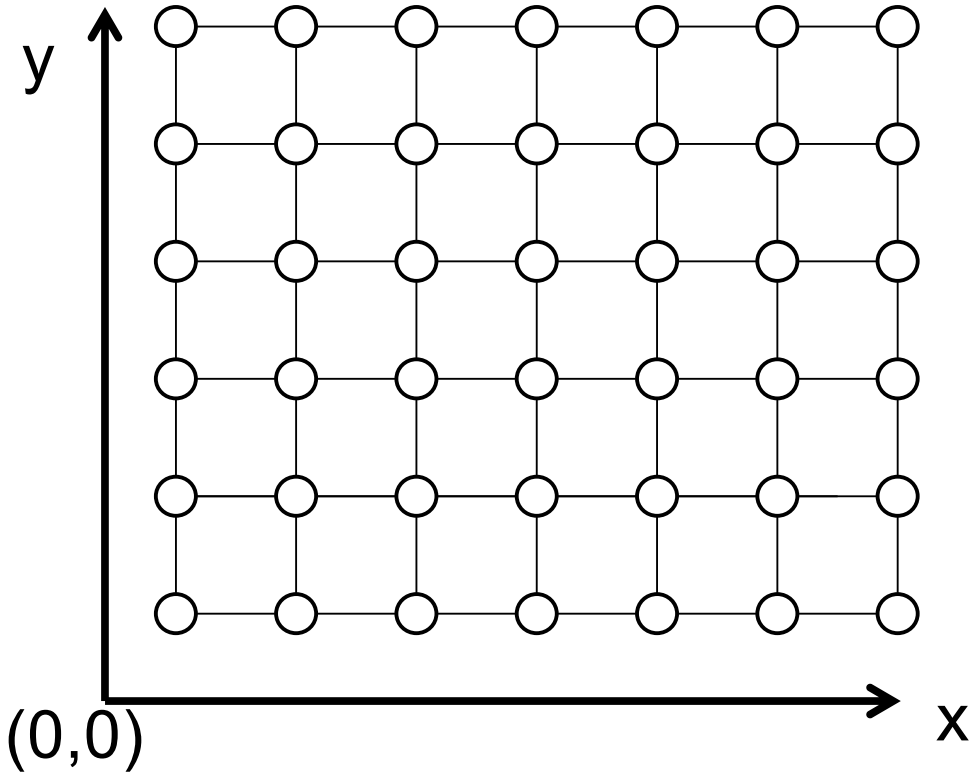
Rasterization

- On a projeté des points sur une image
- = grille de pixels



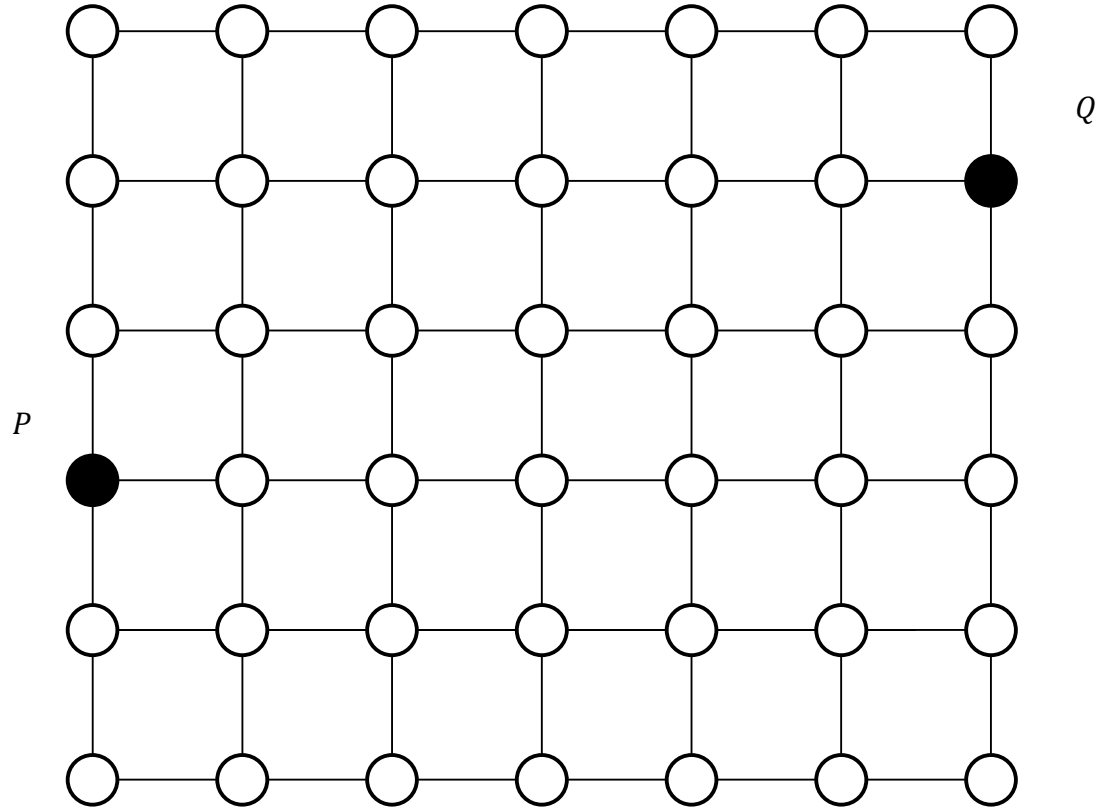
Displays – Pixels

- Pixel: the smallest element of picture
  - Integer position (i,j)
  - Color information (r,g,b)



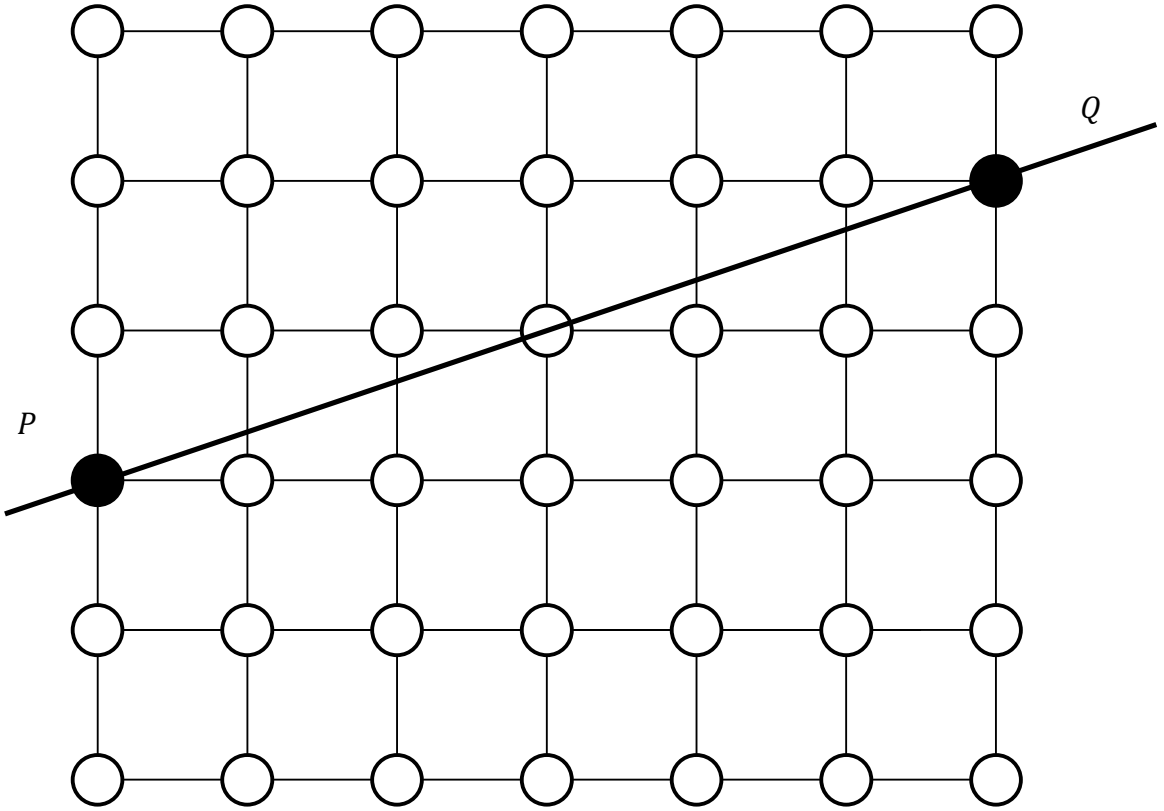
Problem

- Given two points (P, Q) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line



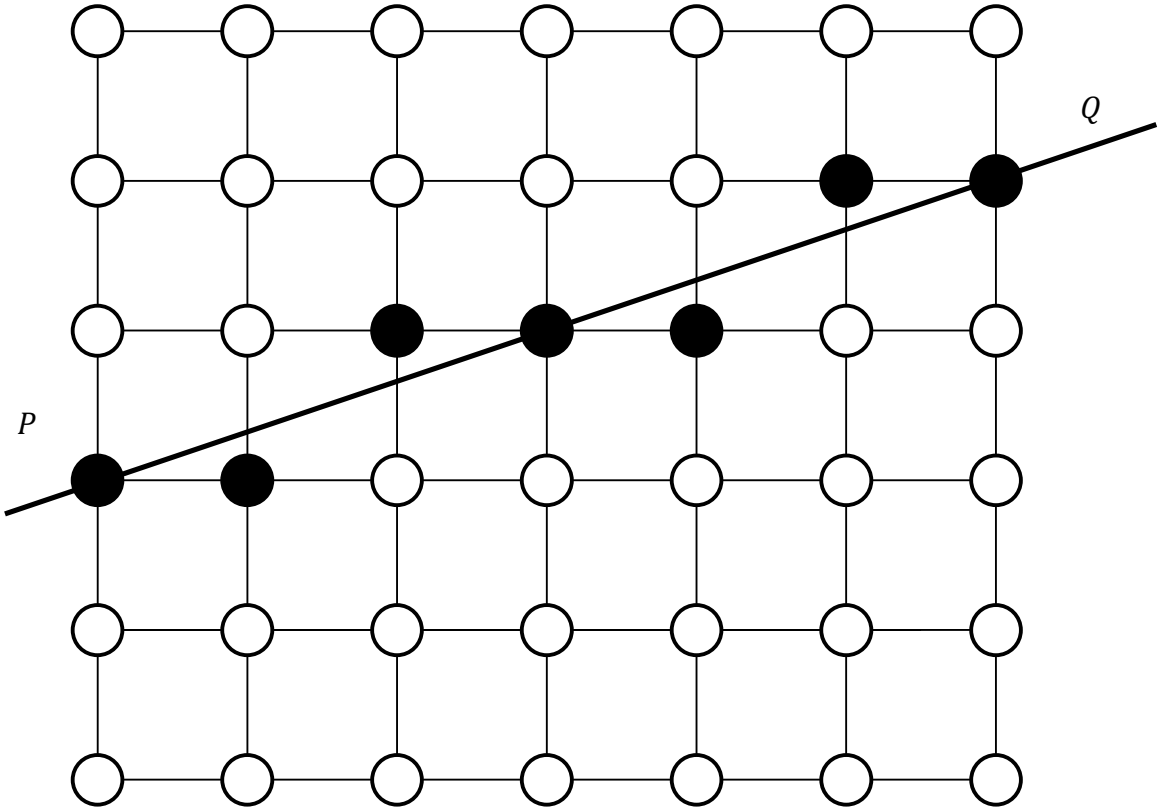
Problem

- Given two points (*P*, *Q*) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line

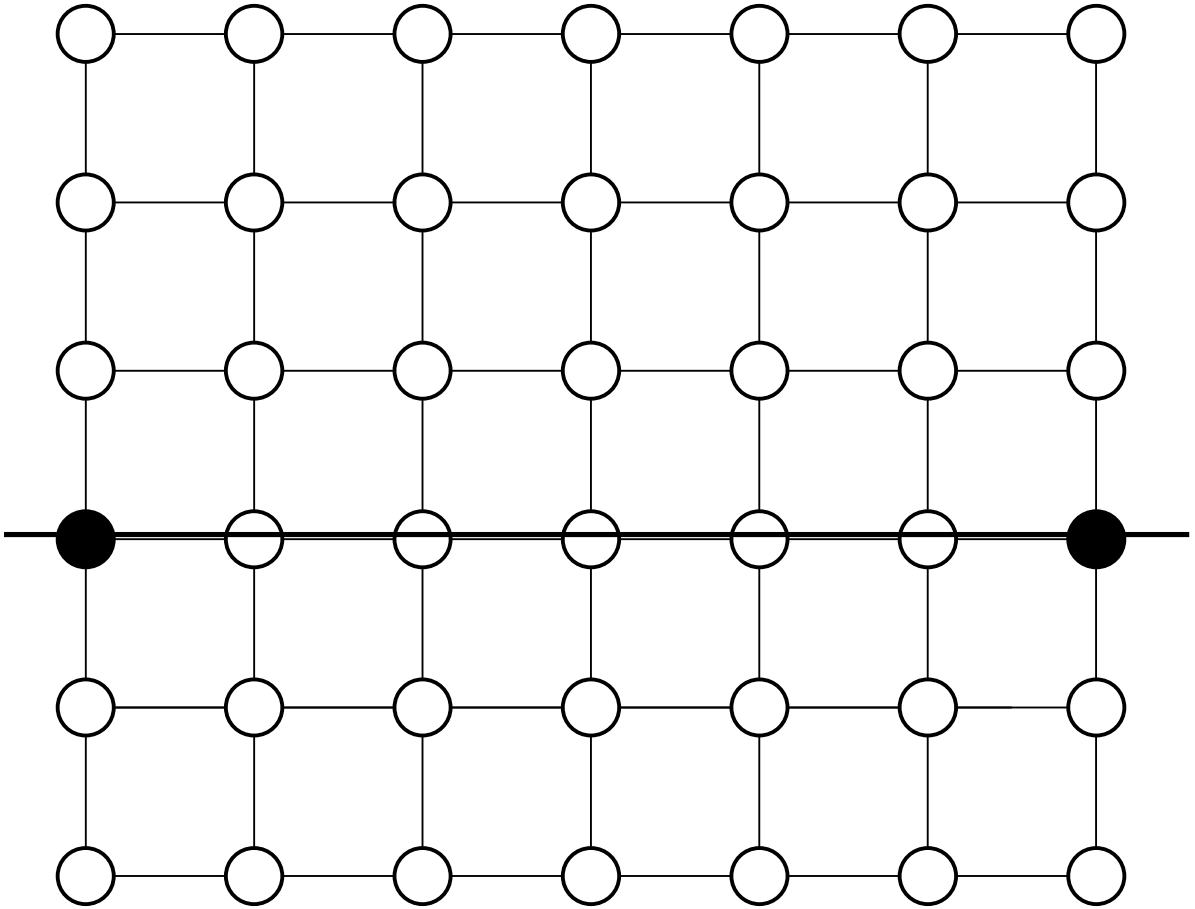


Problem

- Given two points (*P*, *Q*) on the screen (with integer coordinates) determine which pixels should be drawn to display a unit width line

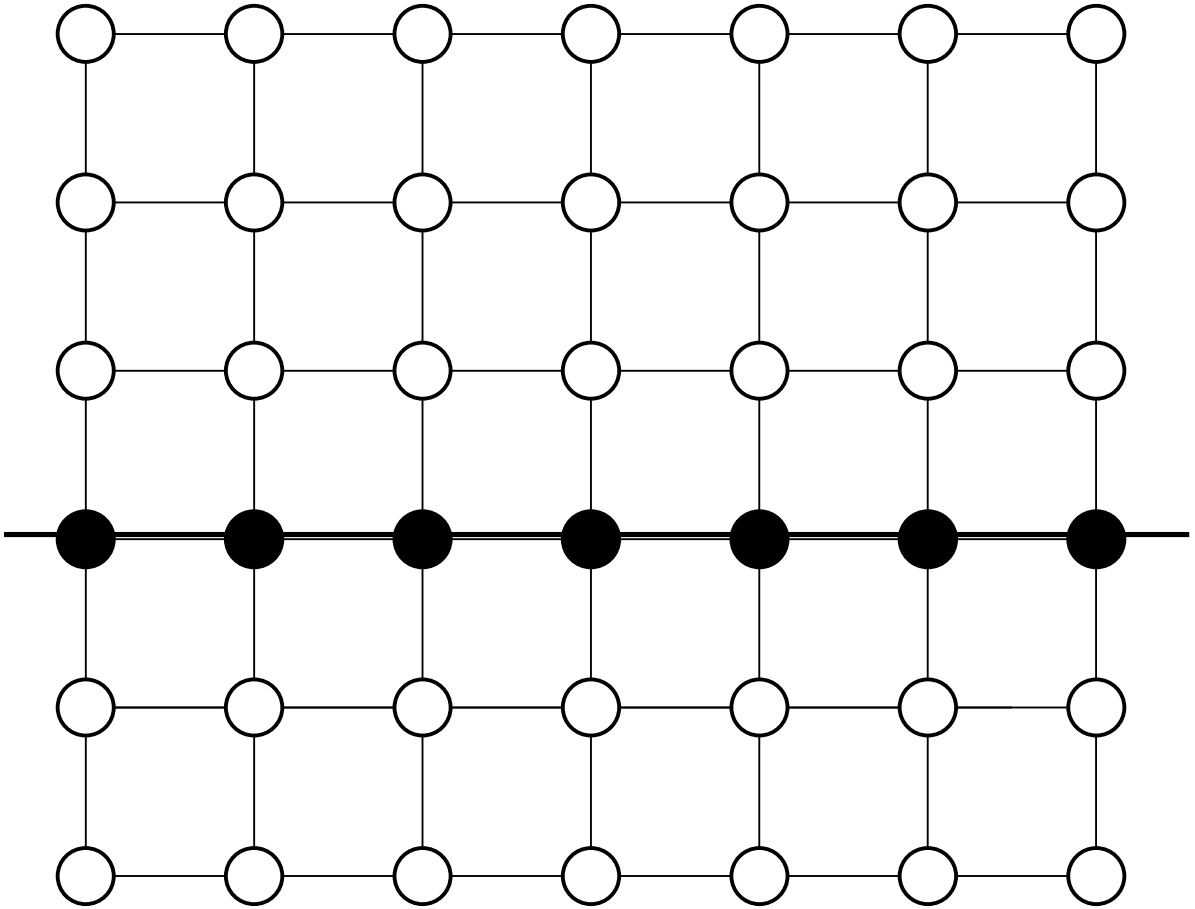


Special Lines - Horizontal

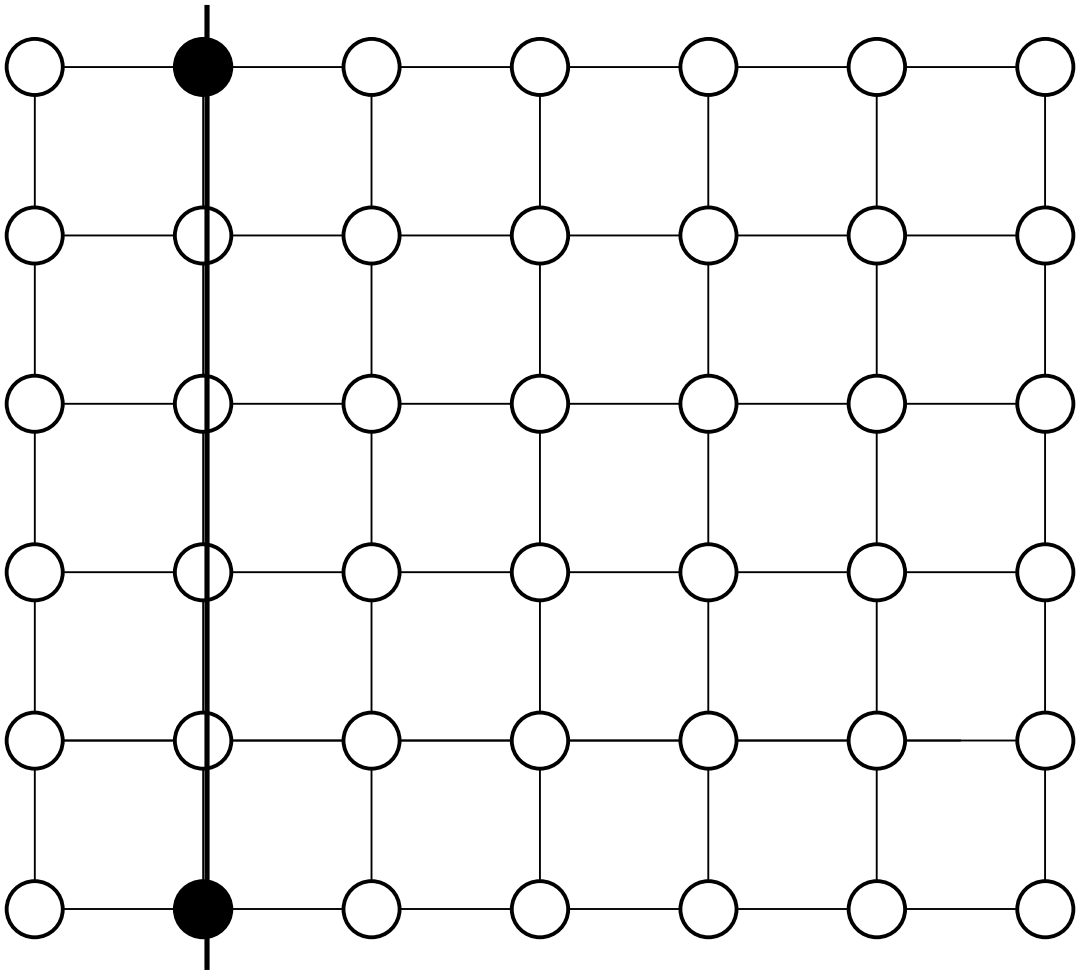


Special Lines - Horizontal

Increment *x* by 1, keep *y* constant

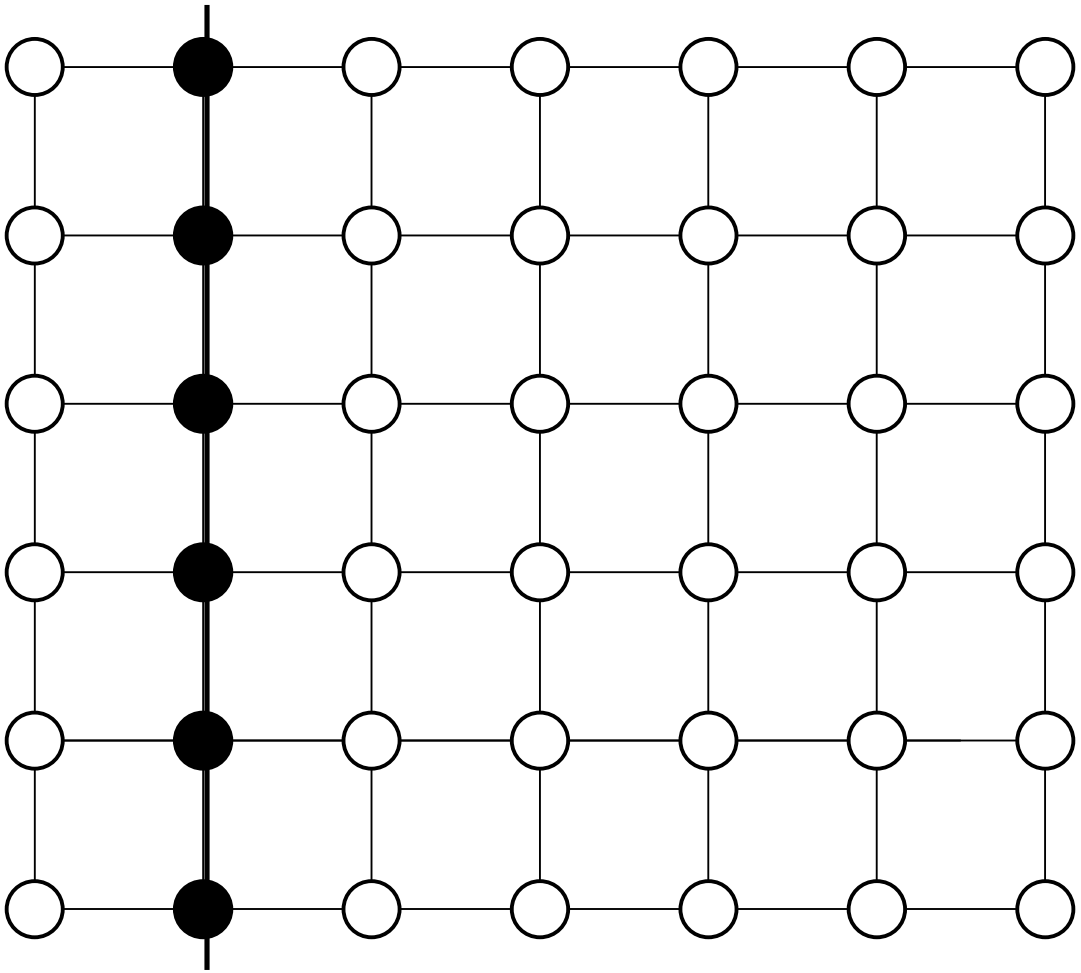


Special Lines - Vertical

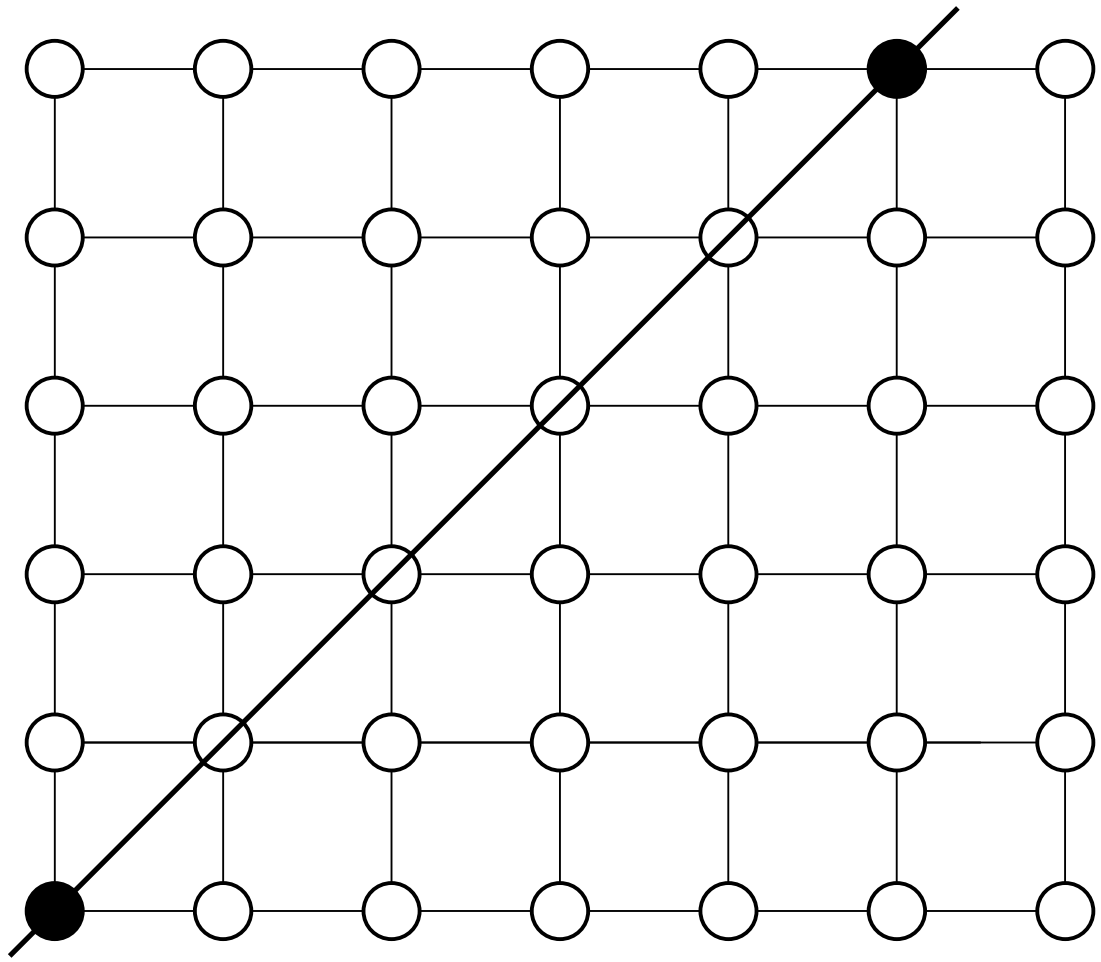


Special Lines - Vertical

Keep *x* constant, increment *y* by 1

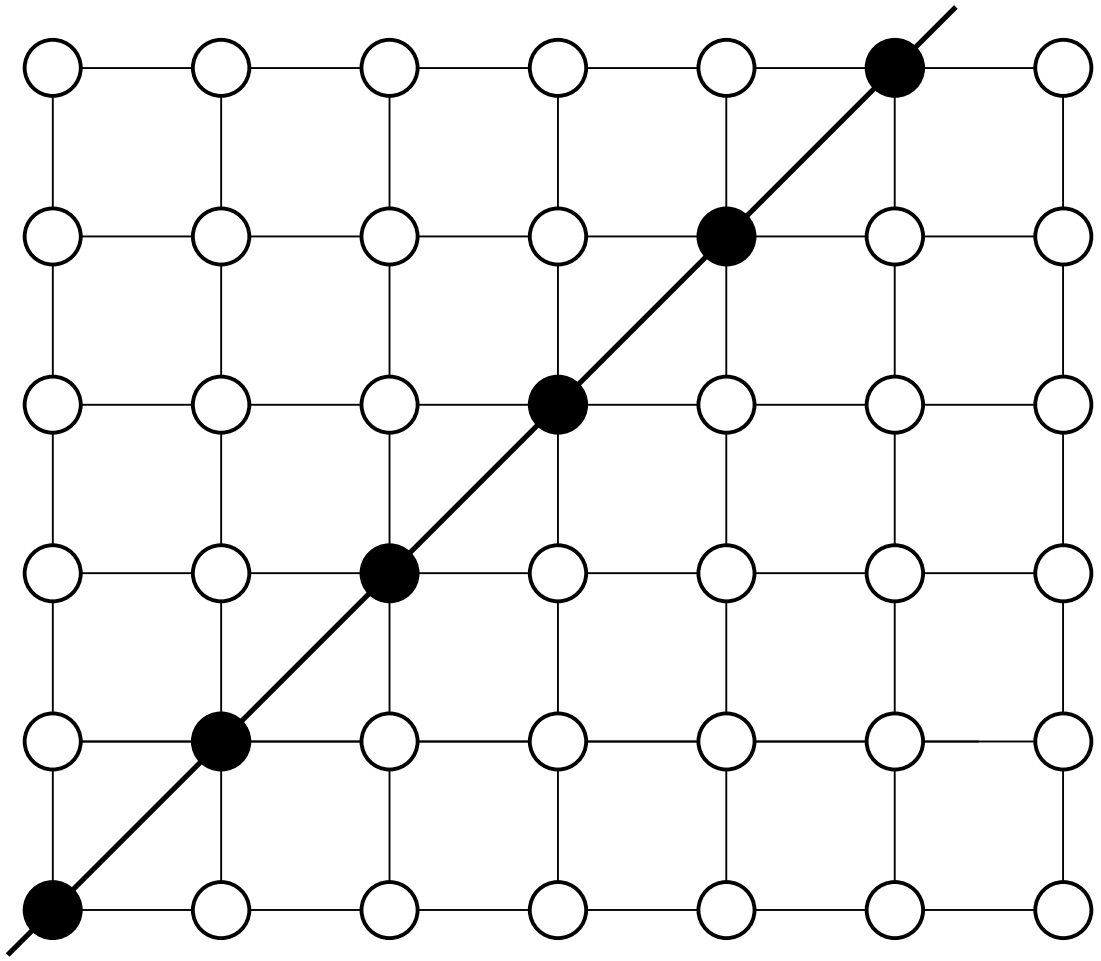


Special Lines - Diagonal

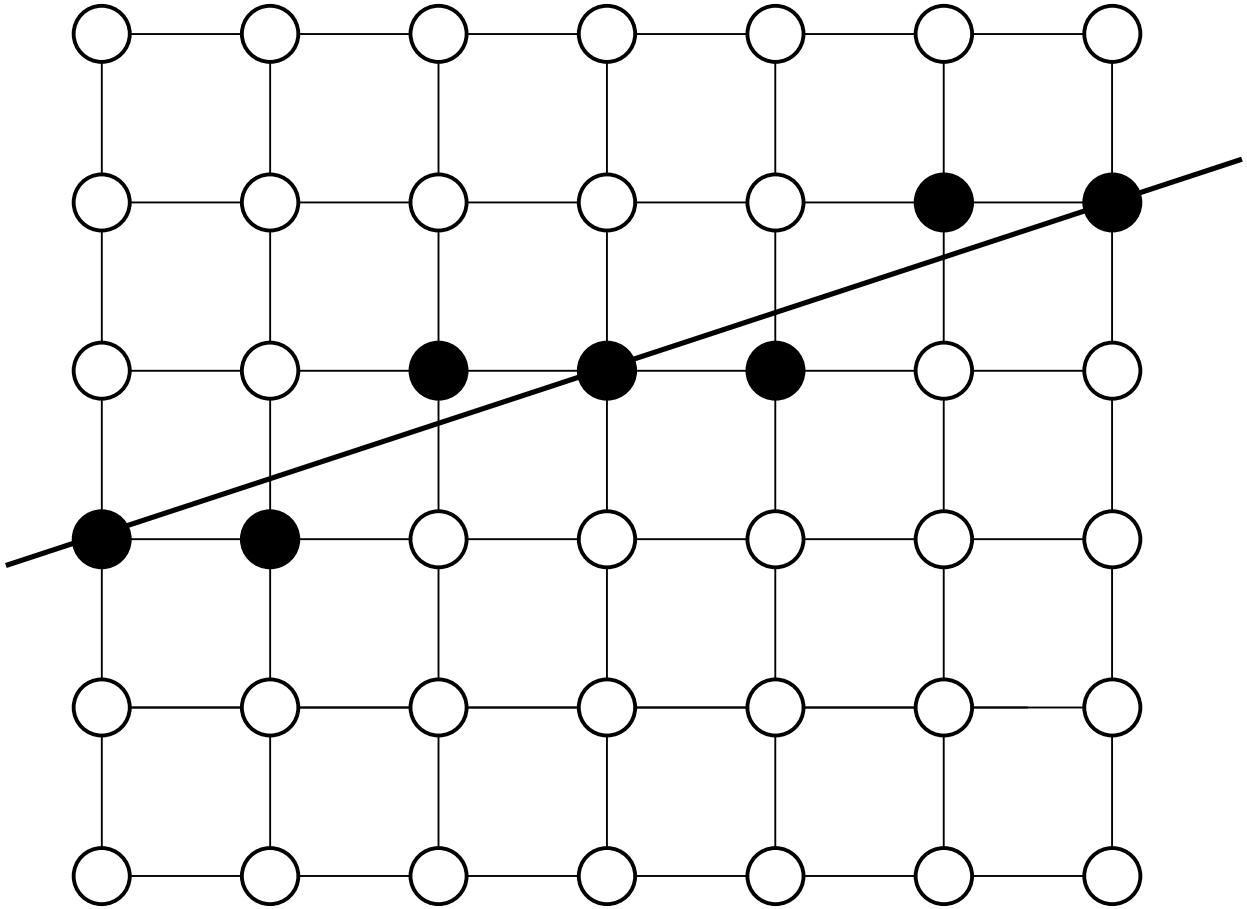


Special Lines - Diagonal

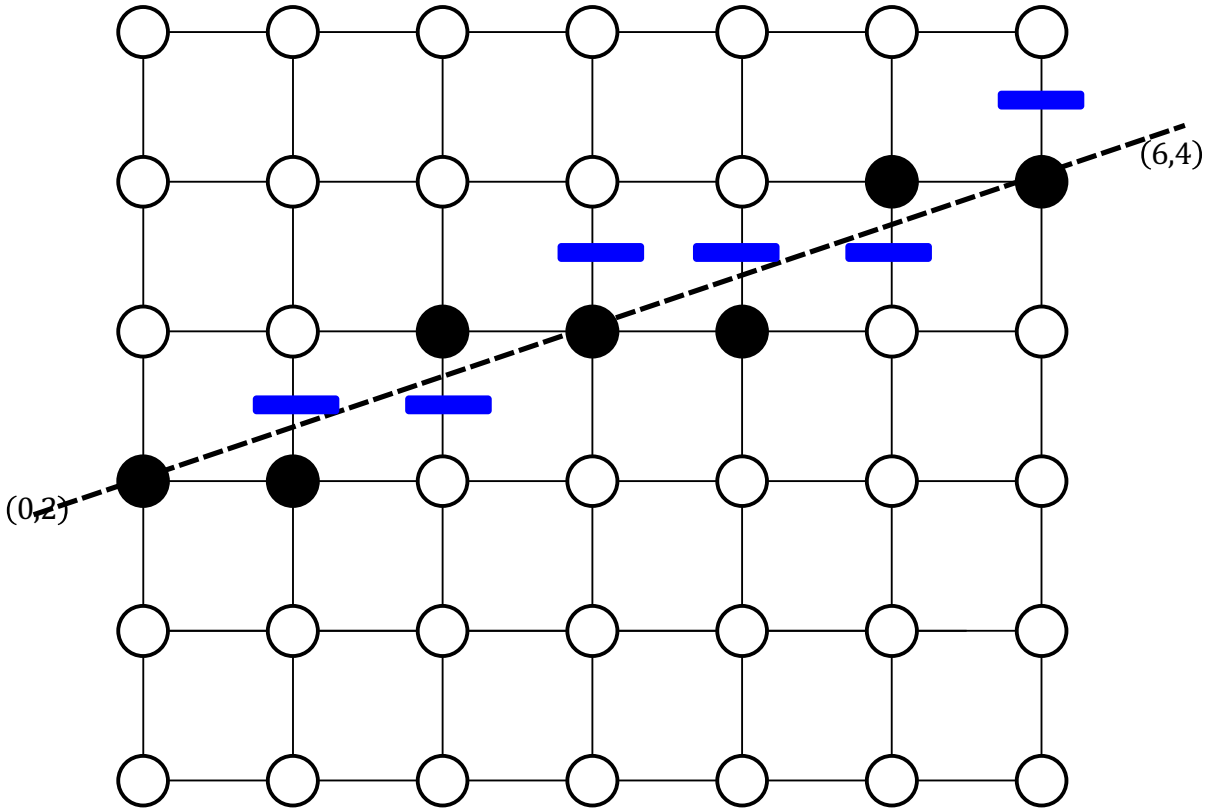
Increment *x* by 1, increment *y* by 1



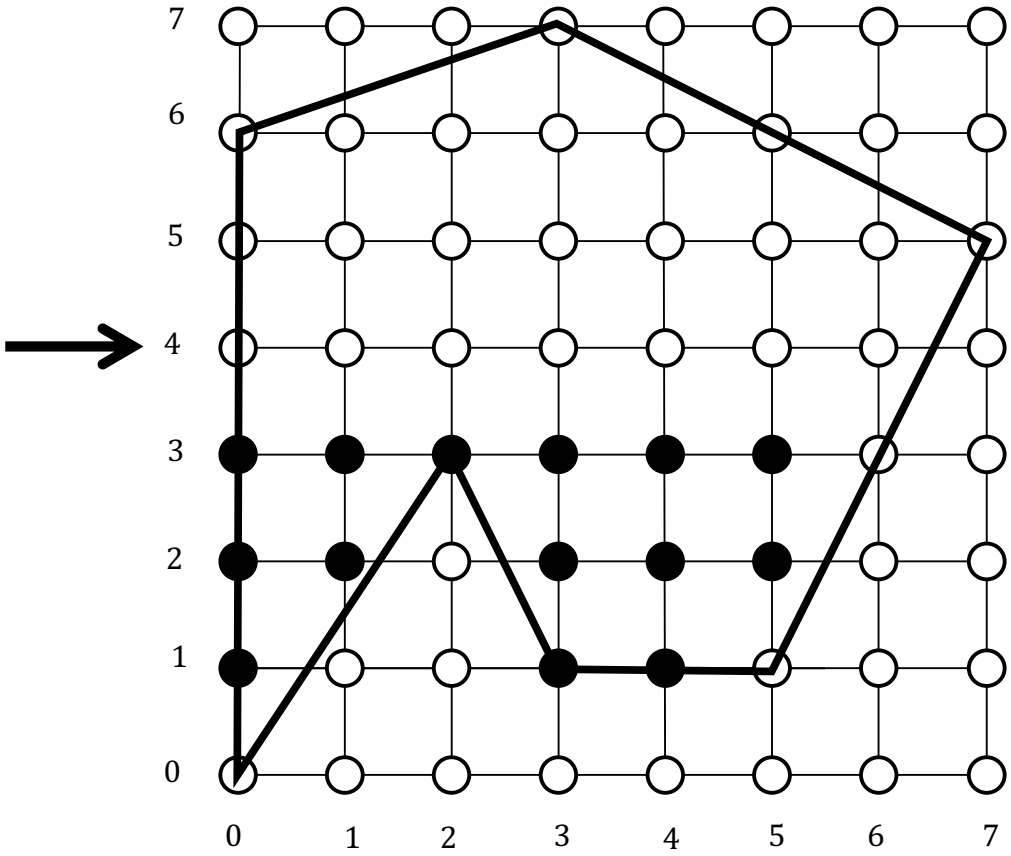
How about Arbitrary Lines?

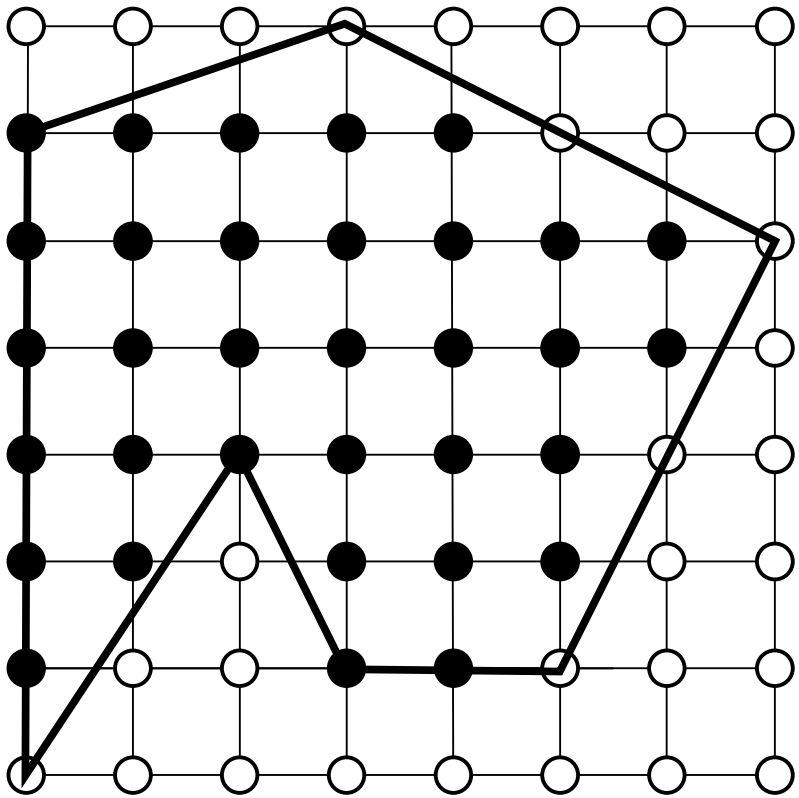


Midpoint Algorithm



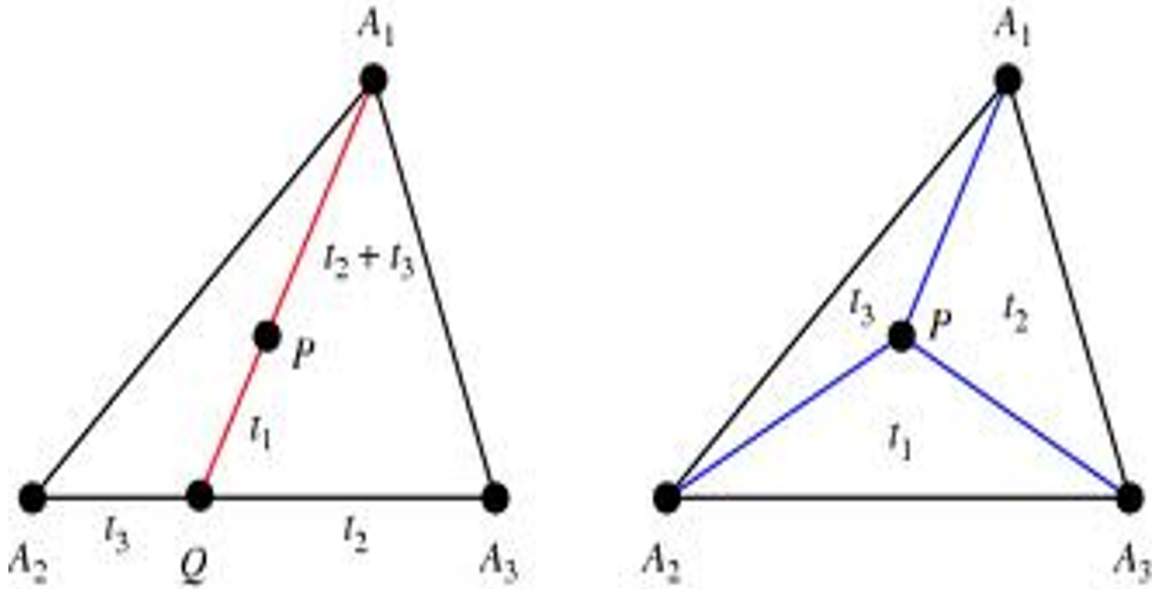
Scan algo for polygon filling





Filling triangles

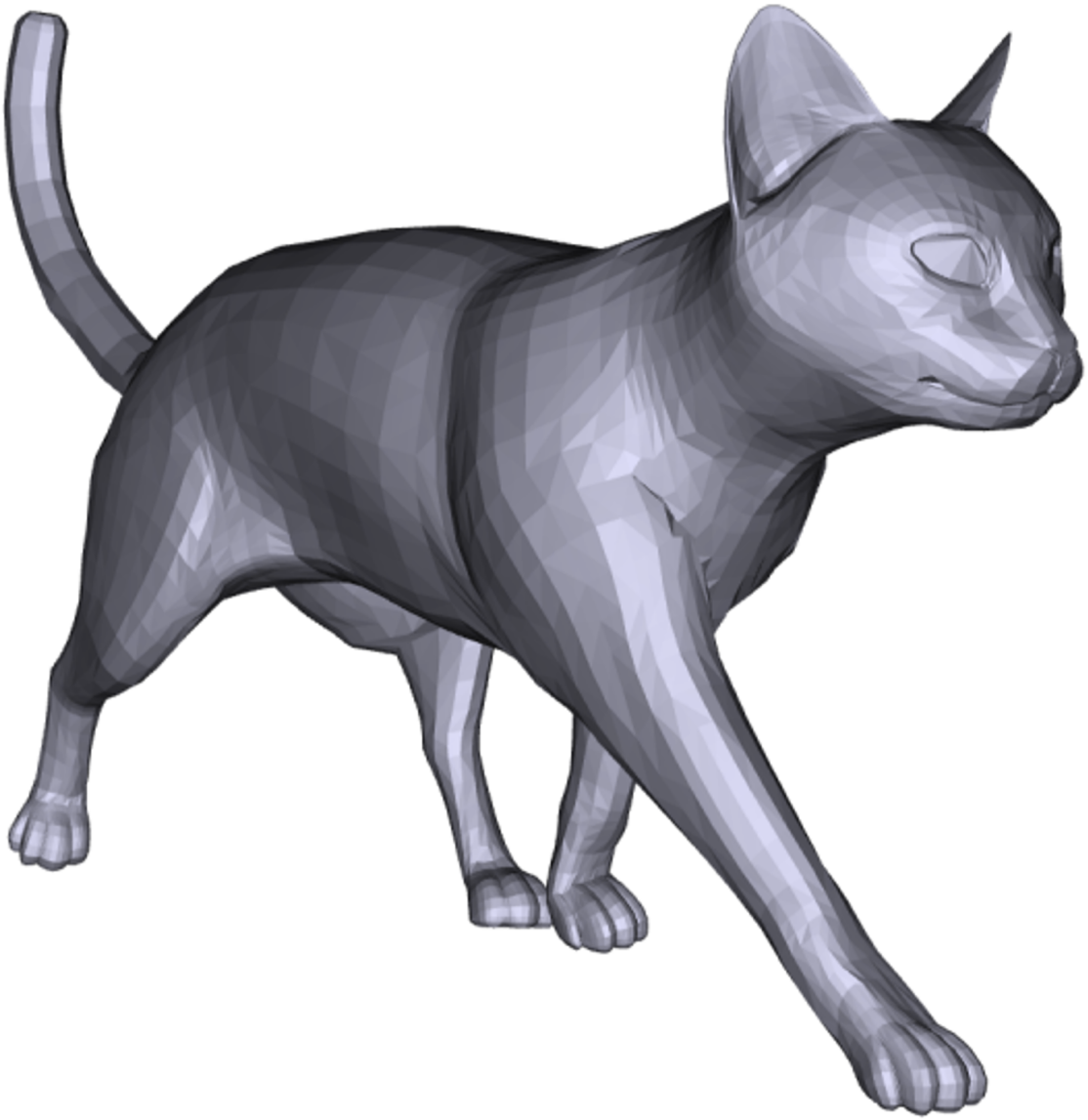
- Test if a point is inside the triangle
- Interpolate the attributes
- Formula?



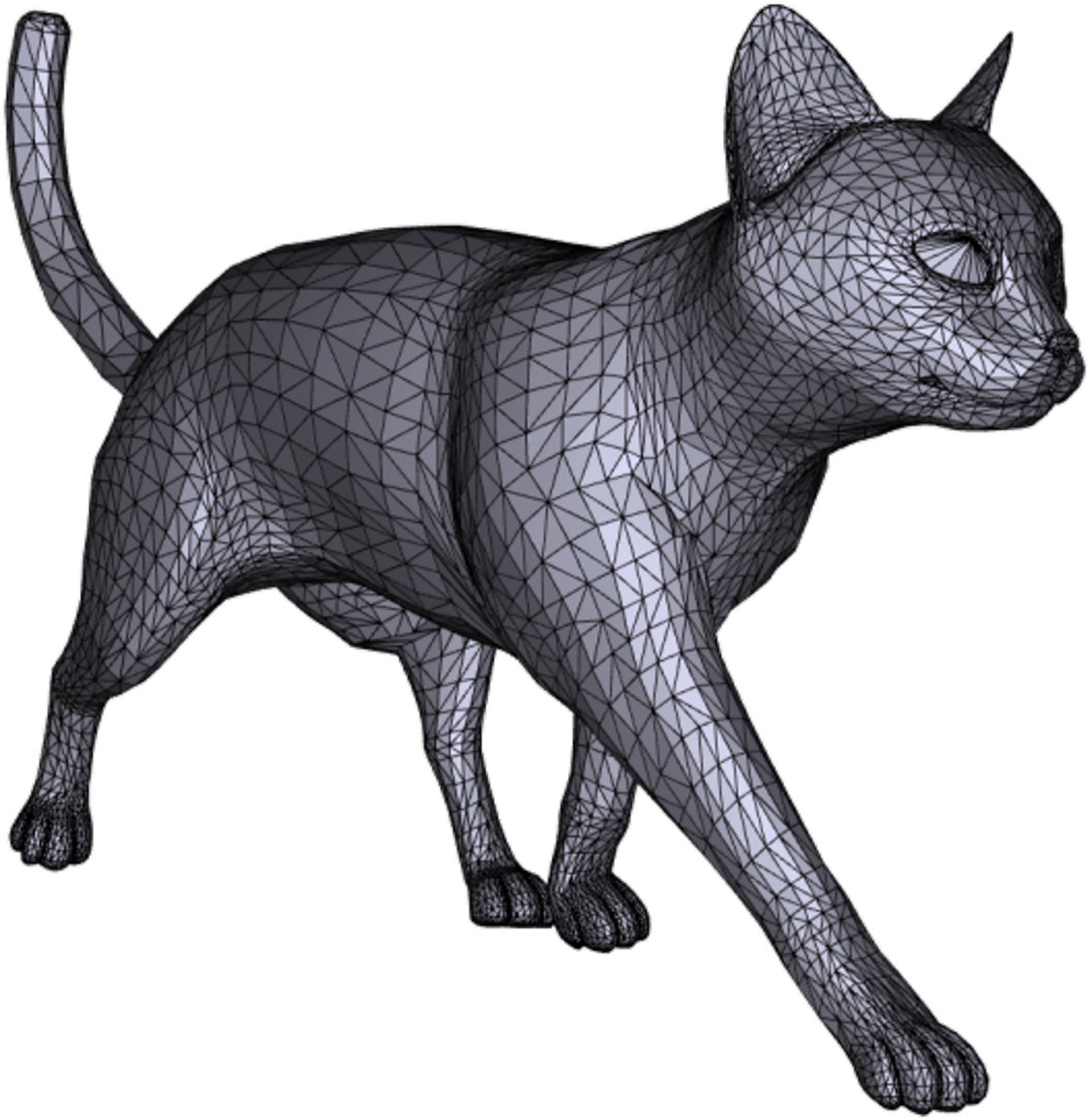
Hidden surface removal

- Back face culling
- Frustum culling
- Occlusion

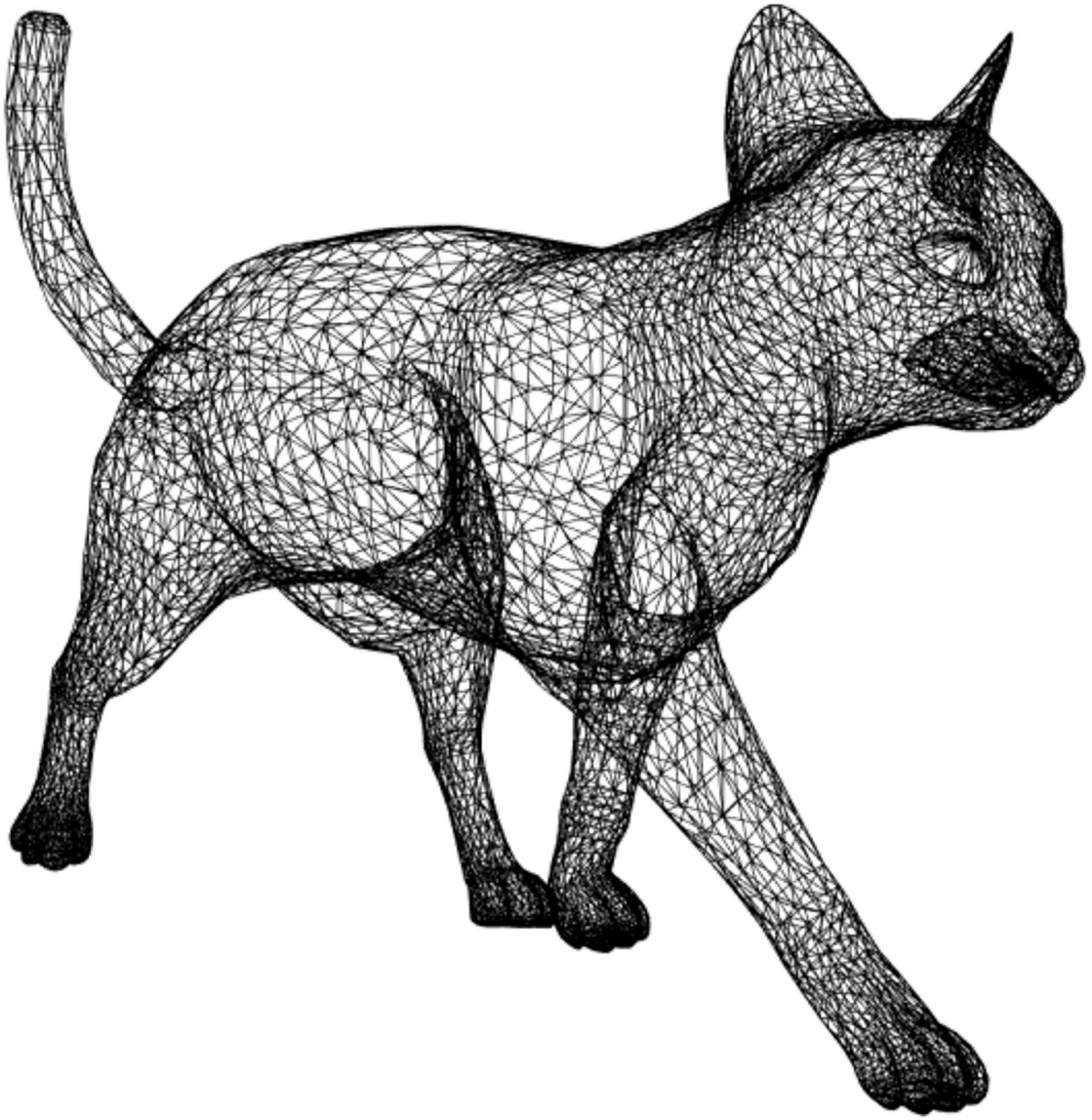
Hidden Surfaces



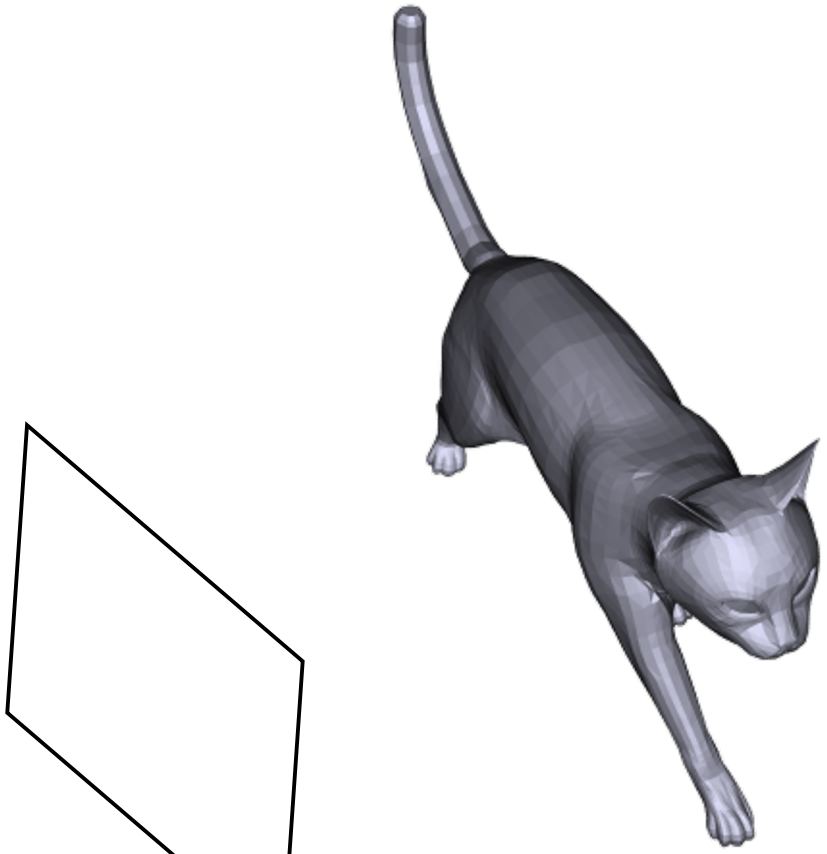
Hidden Surfaces



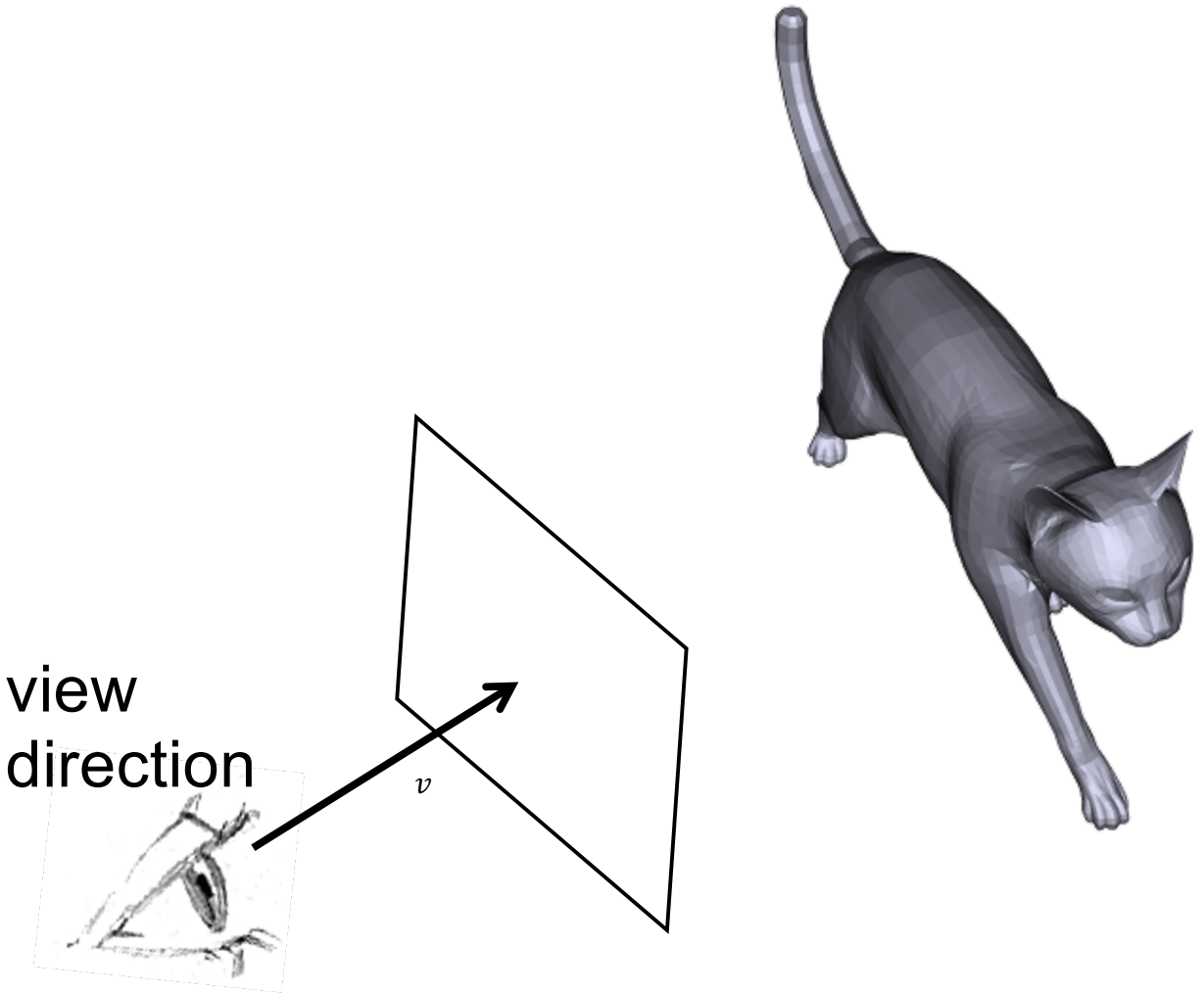
Hidden Surfaces



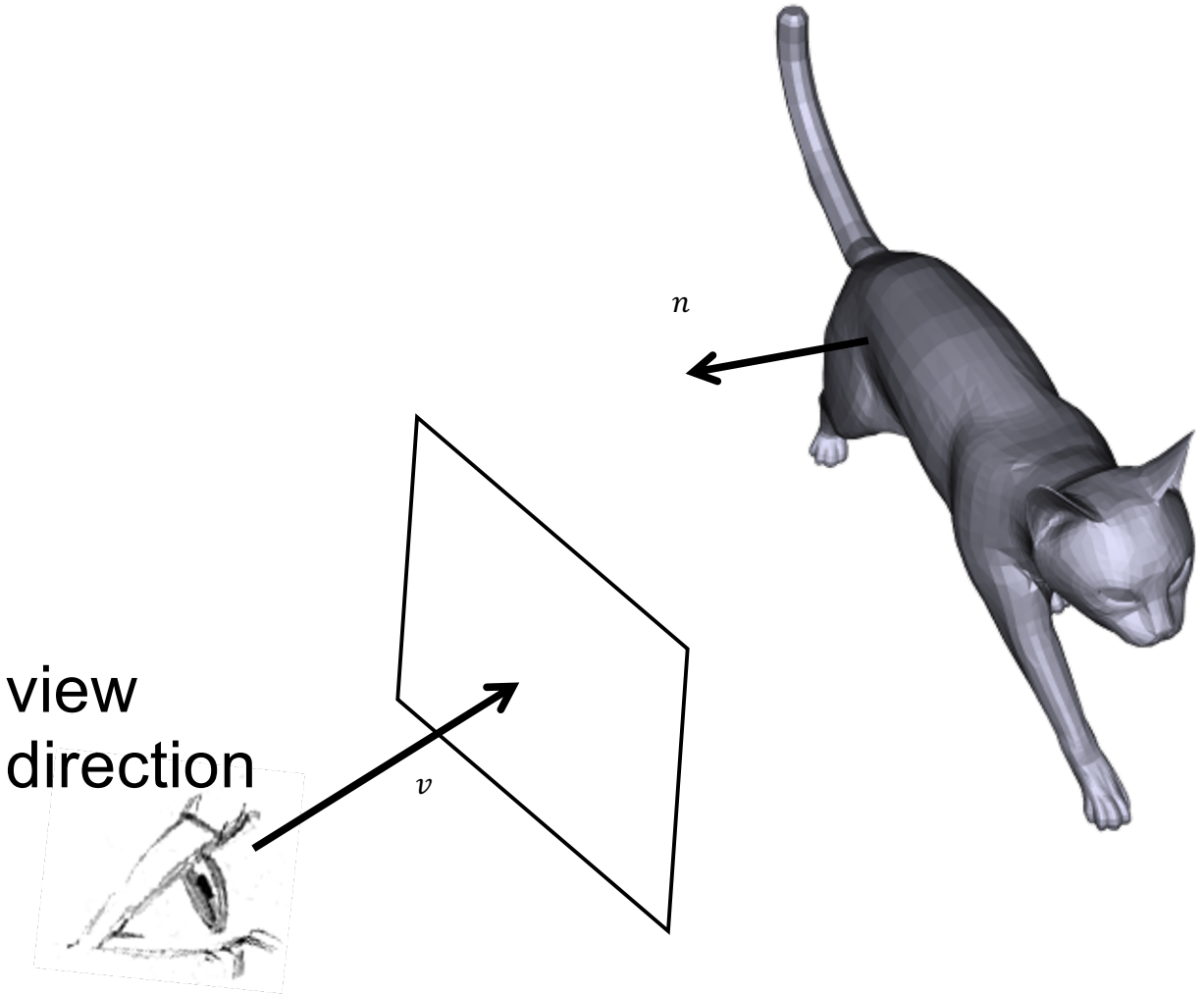
Backface Culling



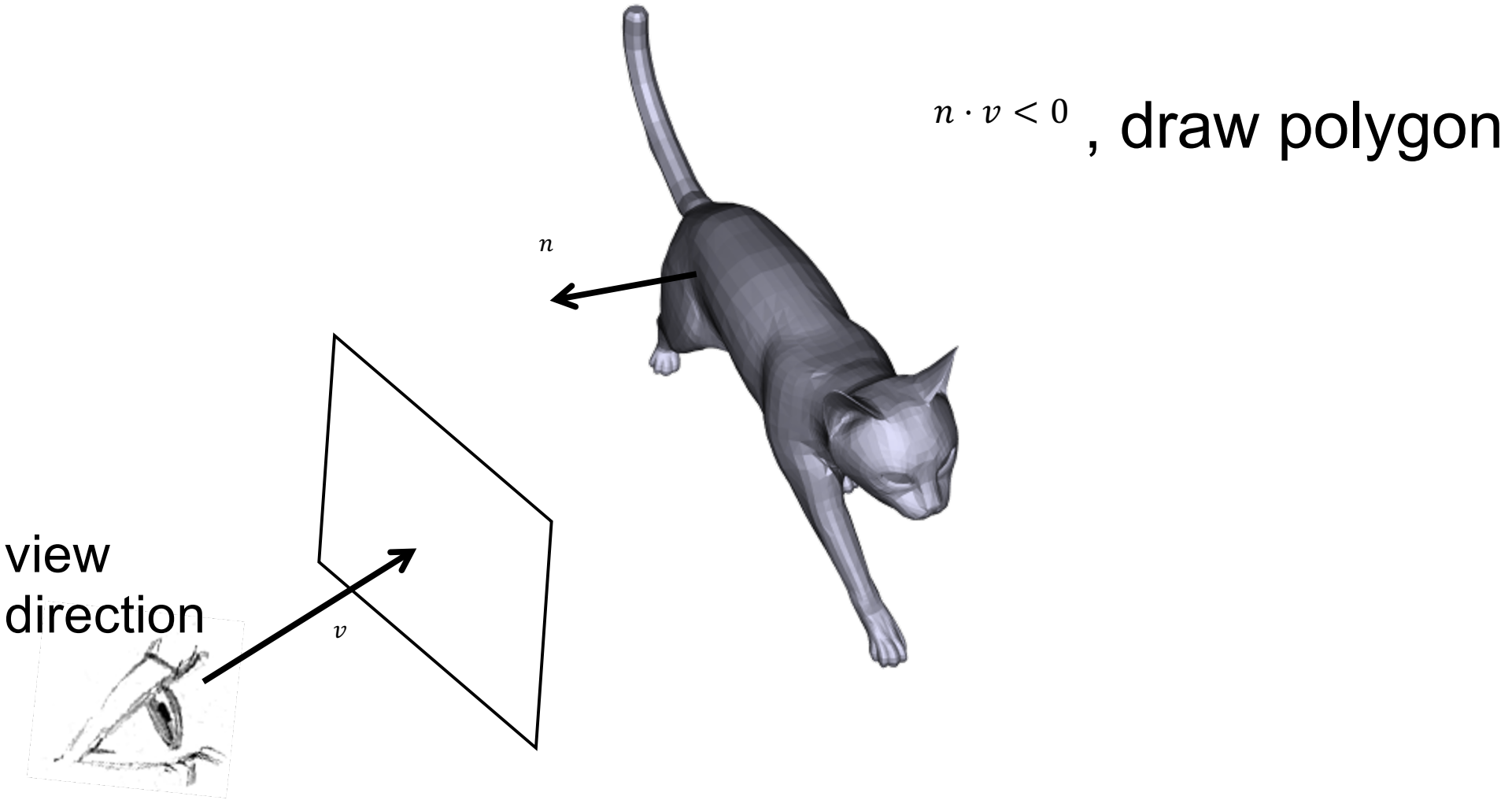
Backface Culling



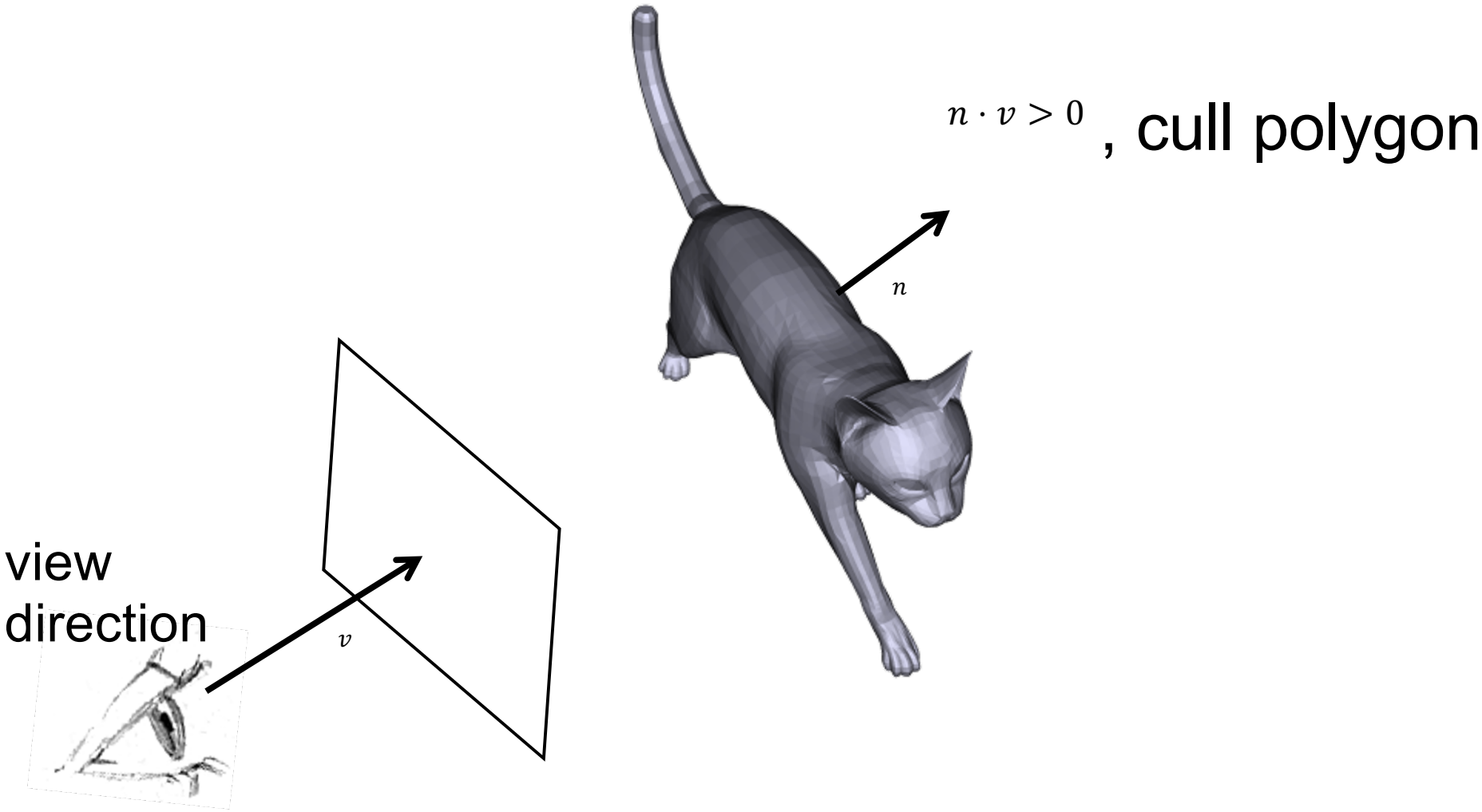
Backface Culling



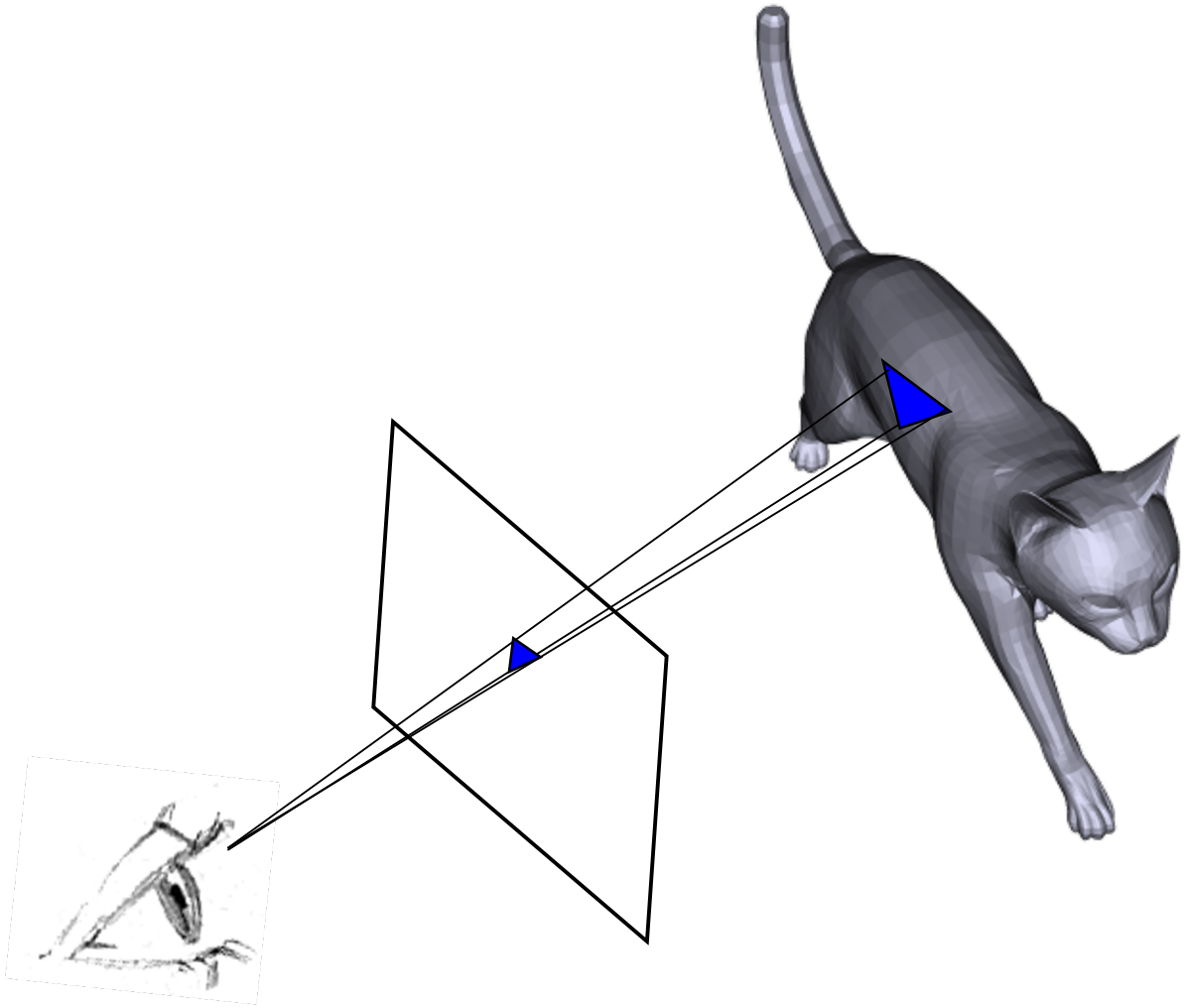
Backface Culling



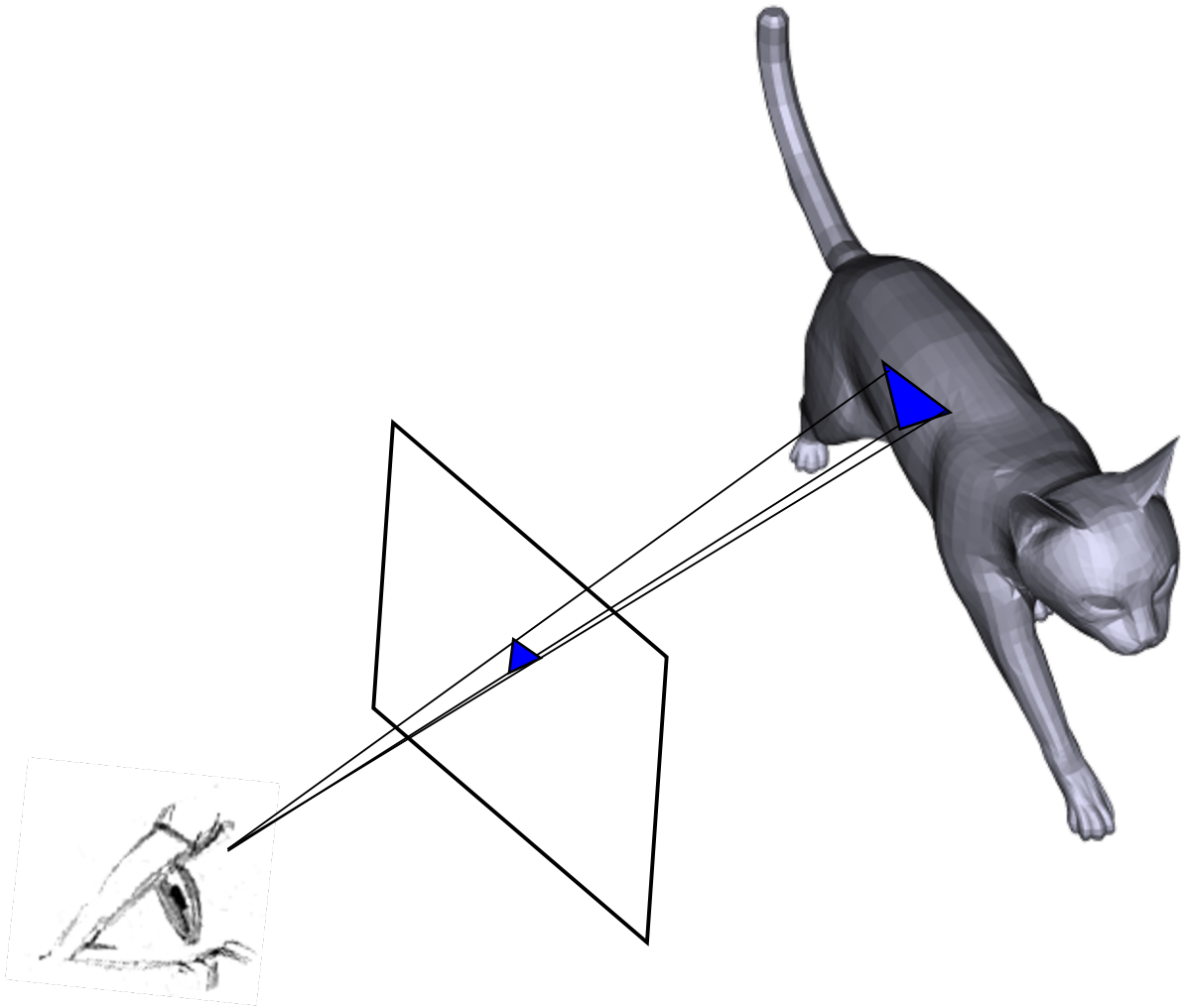
Backface Culling

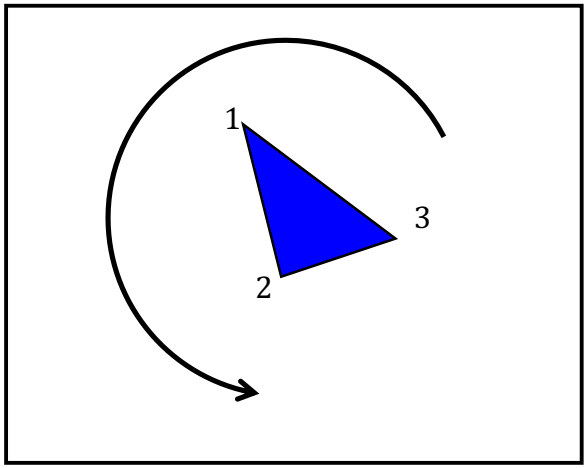


Backface Culling



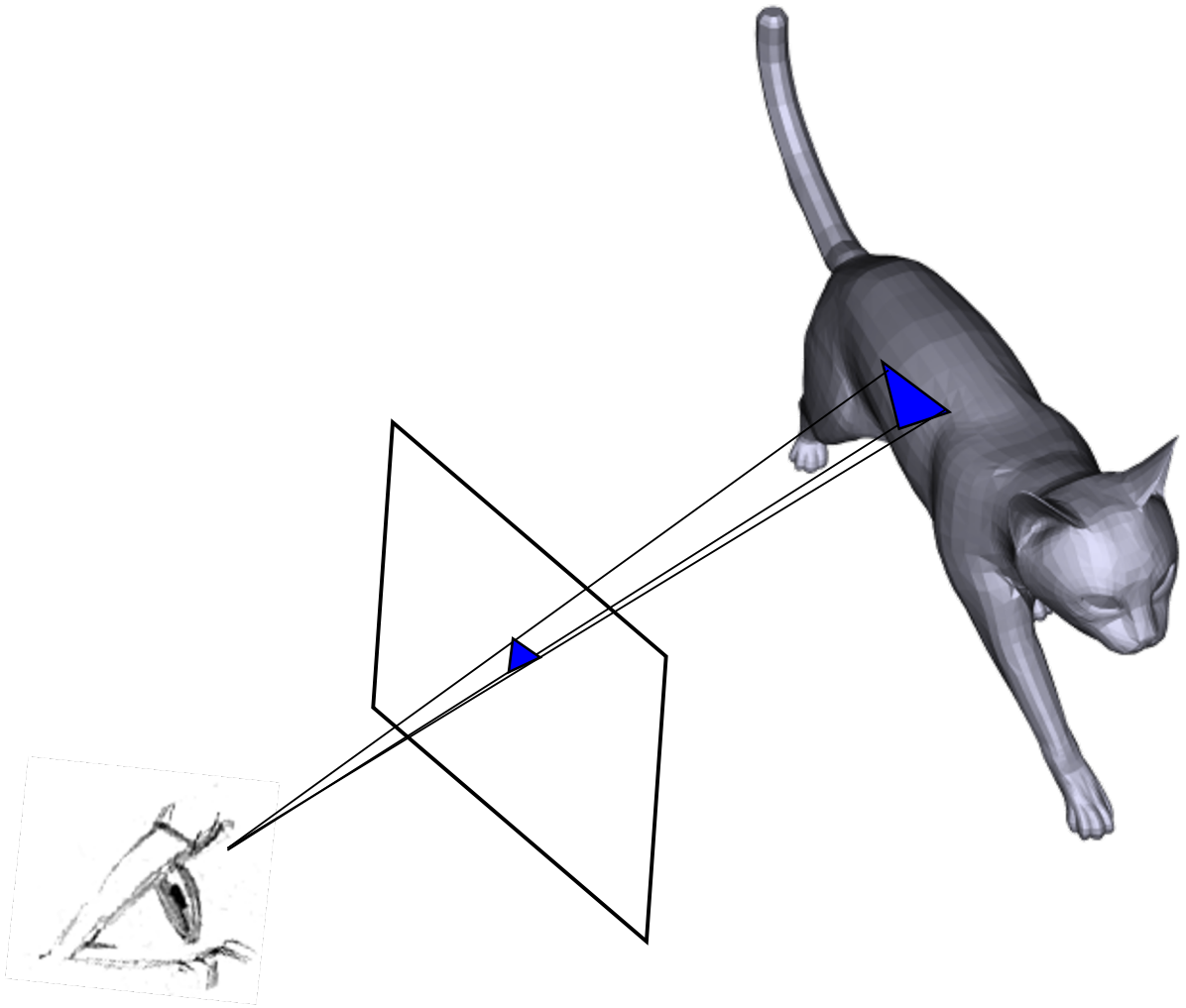
Backface Culling

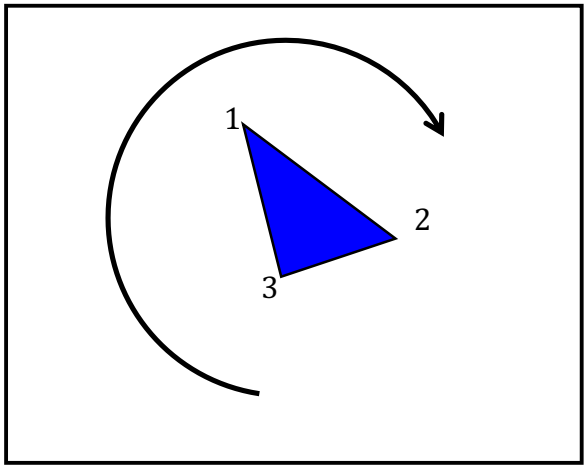




counter clock-wise orientation, draw polygor

Backface Culling





clock-wise orientation,
cull polygon

Backface Culling

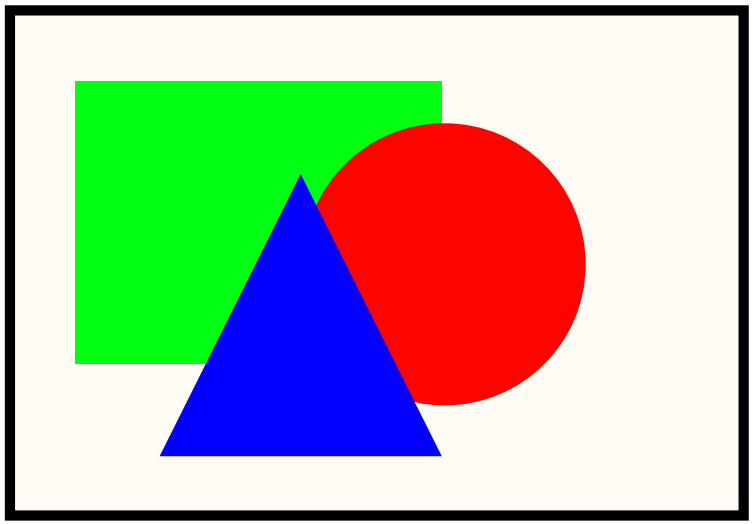
- Advantages
- Improves rendering speed by removing roughly half of polygons from scan conversion
- Disadvantages
- Assumes closed surface with consistently oriented polygons

Backface Culling

- Is this all we have to do?

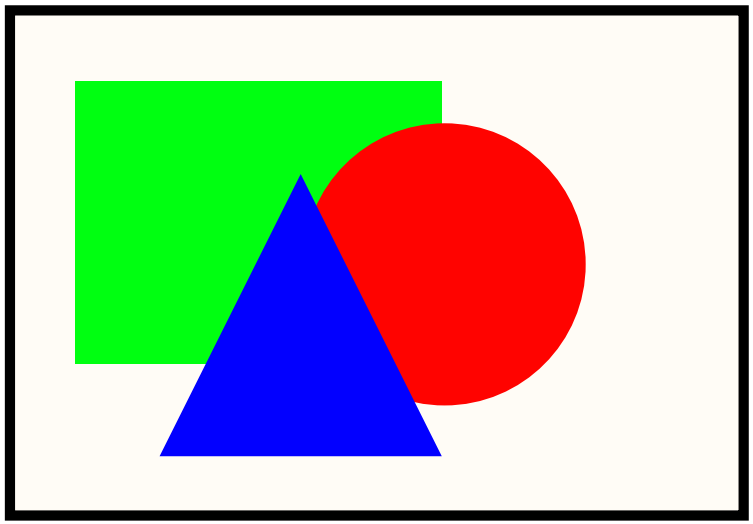
Backface Culling

- Is this all we have to do? No!
  - Can still have 2 (or more) front faces that map to the same screen pixel



Backface Culling

- Is this all we have to do? No!
  - Can still have 2 (or more) front faces that map to the same screen pixel
  - Which actually gets drawn?



Depth (“Z”) Buffer

- Simple modification to scan-conversion
- Maintain a separate buffer storing the closest “z” value for each pixel
- Only draw pixel if depth value is closer than stored “z” value
- Update buffer with closest depth value

Depth (“Z”) Buffer

- Advantages
- Simple to implement
- Disadvantages
- Requires extra storage space
- Still lots of overdraw

==> Implementation for your renderer...