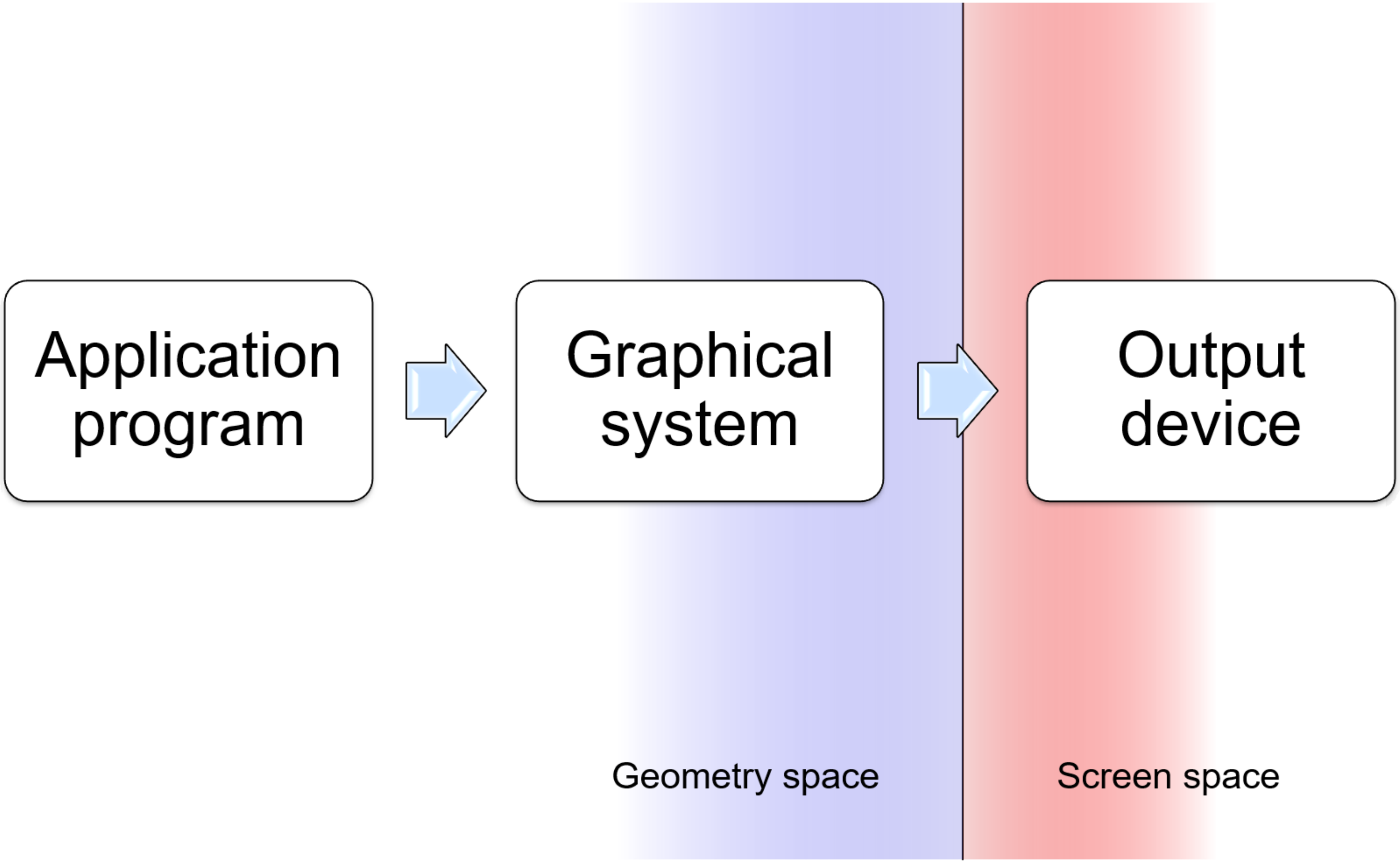
Fundamentals of Computer Graphics and Image Processing

Rasterization (03)

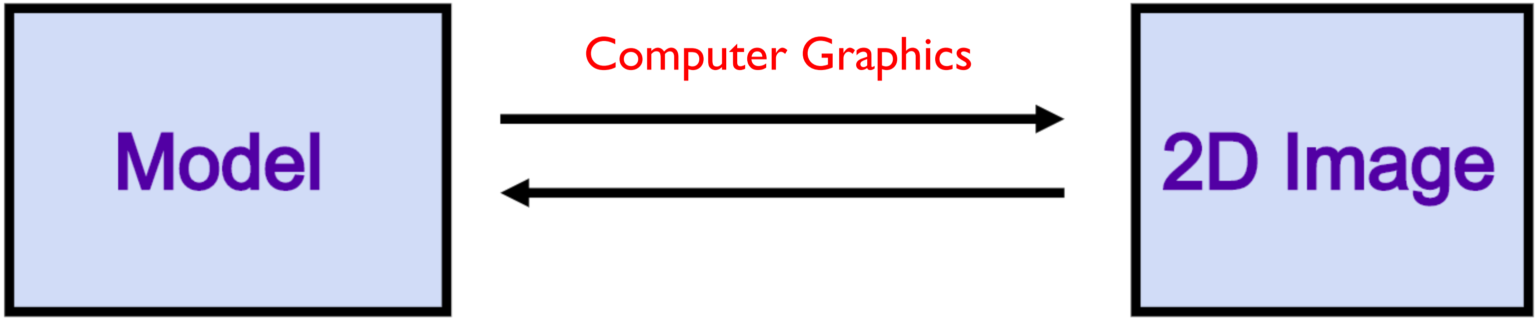
doc. RNDr. Martin Madaras, PhD.
martin.madaras@fmph.uniba.sk

Last lessons summary

CG reference model

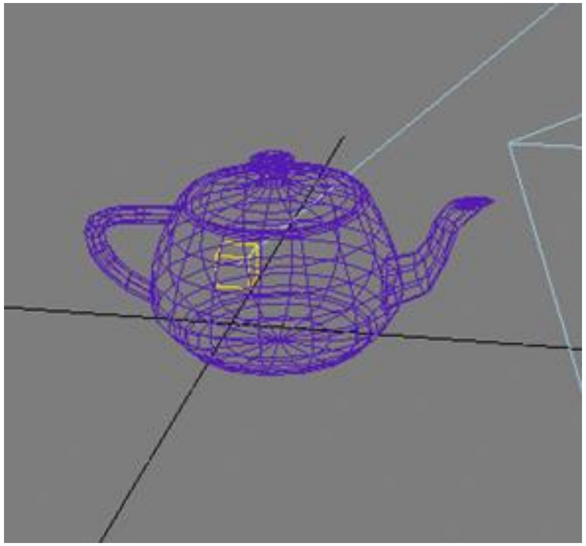


Computer Vision/ Computer Graphics

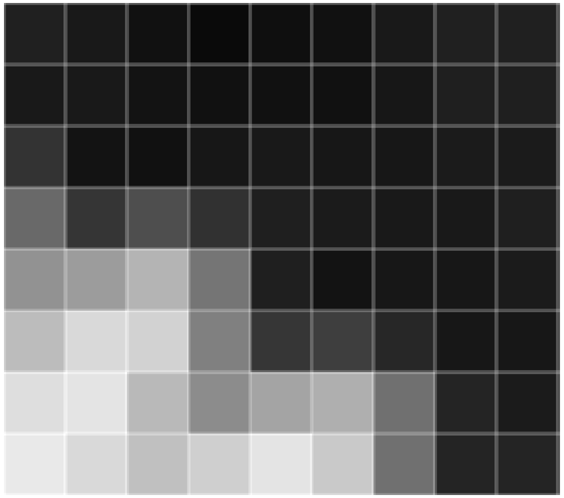


CG reference model

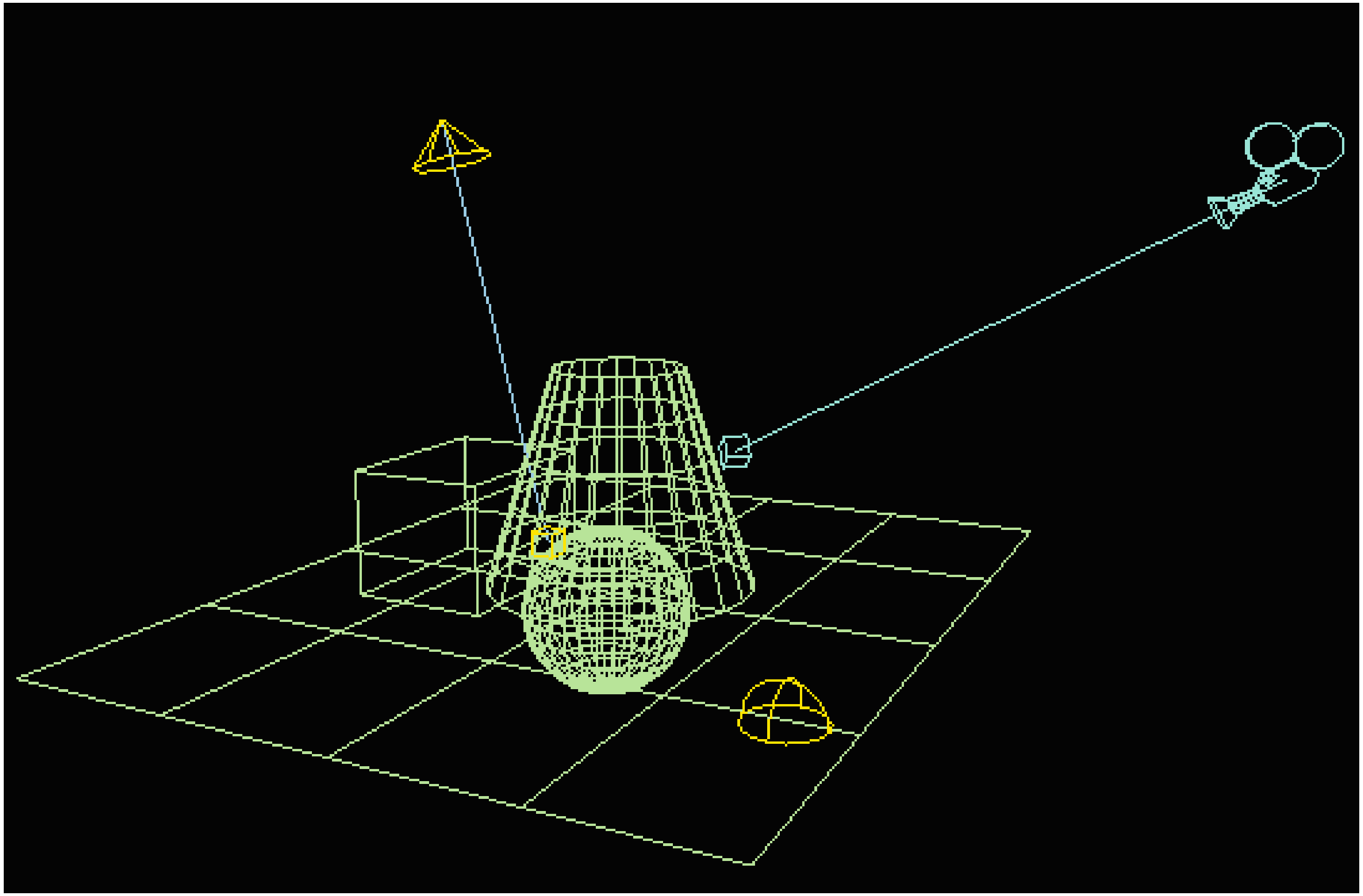
- Geometry space
  - continuous
  - 3Dimensional



- Screen space
  - discrete
  - 2Dimensional

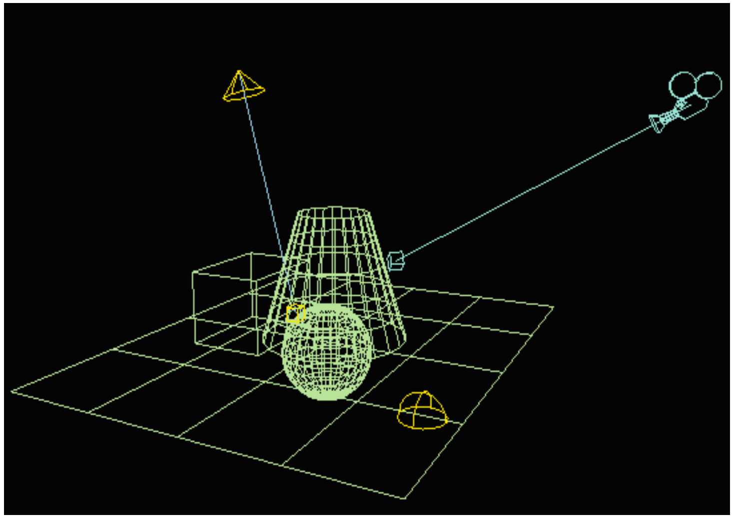


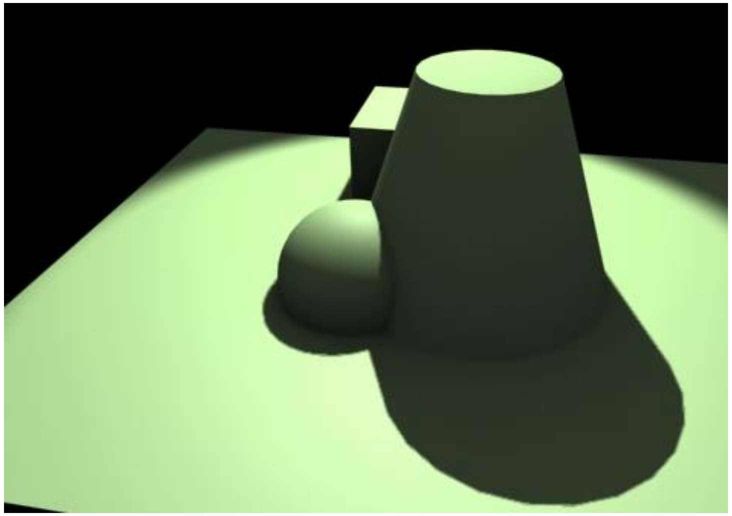
3D Scene vs. 2D image



Geometry vs. screen space

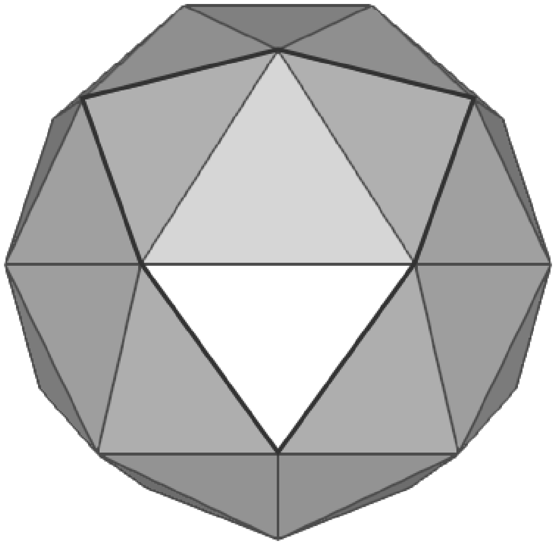
3D	2D
Continuous	Discrete
Parametric	Non-parametric
Models	Pixels





3D polygon rendering

- Many applications use rendering of 3D polygons with direct illumination



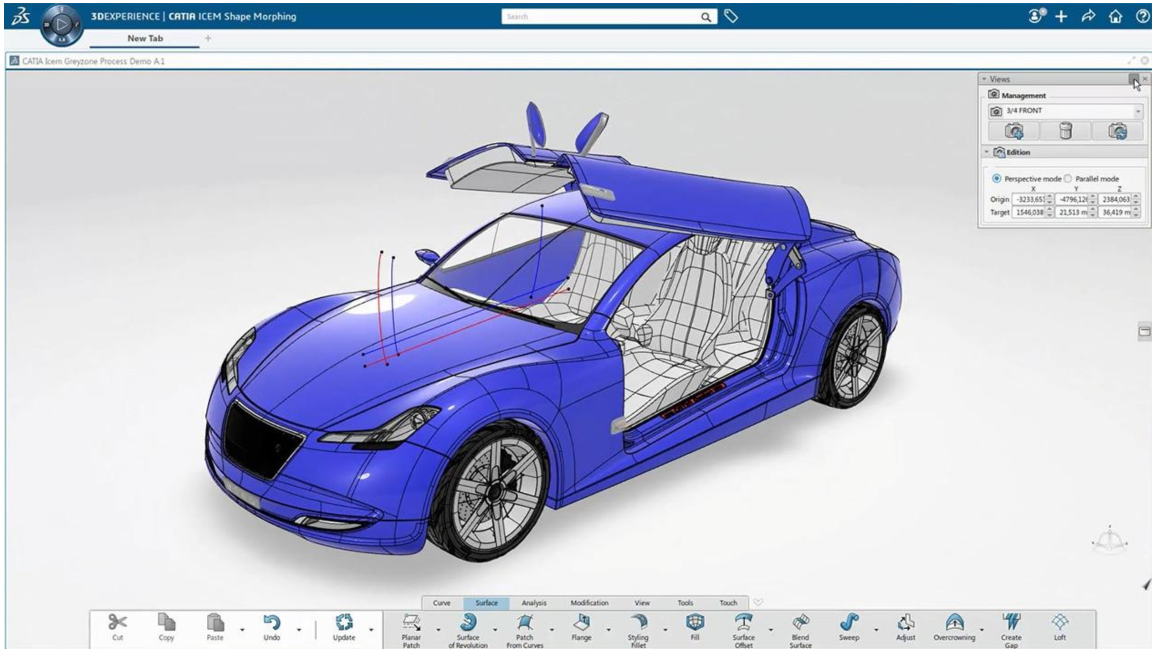
3D polygon rendering

- Many applications use rendering of 3D polygons with direct illumination

Quake 3, ID software

3D polygon rendering

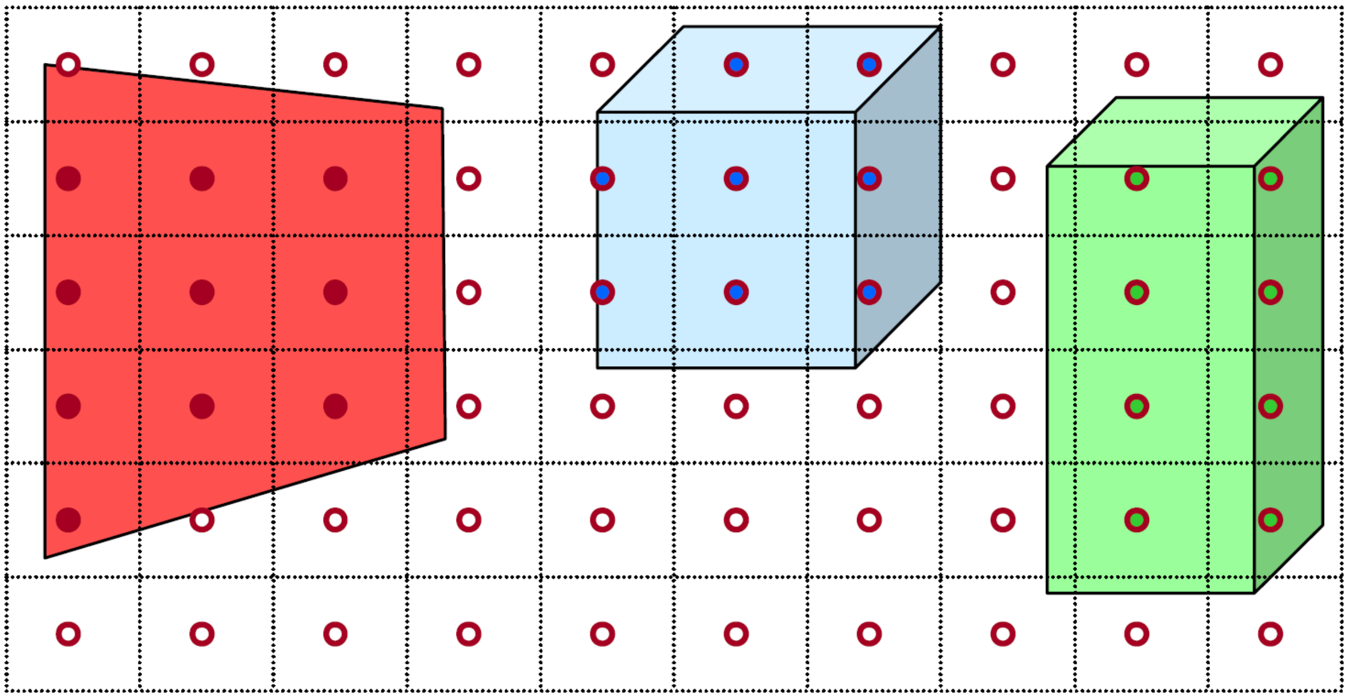
- Many applications use rendering of 3D polygons with direct illumination



CATIA, Dassault Systemes

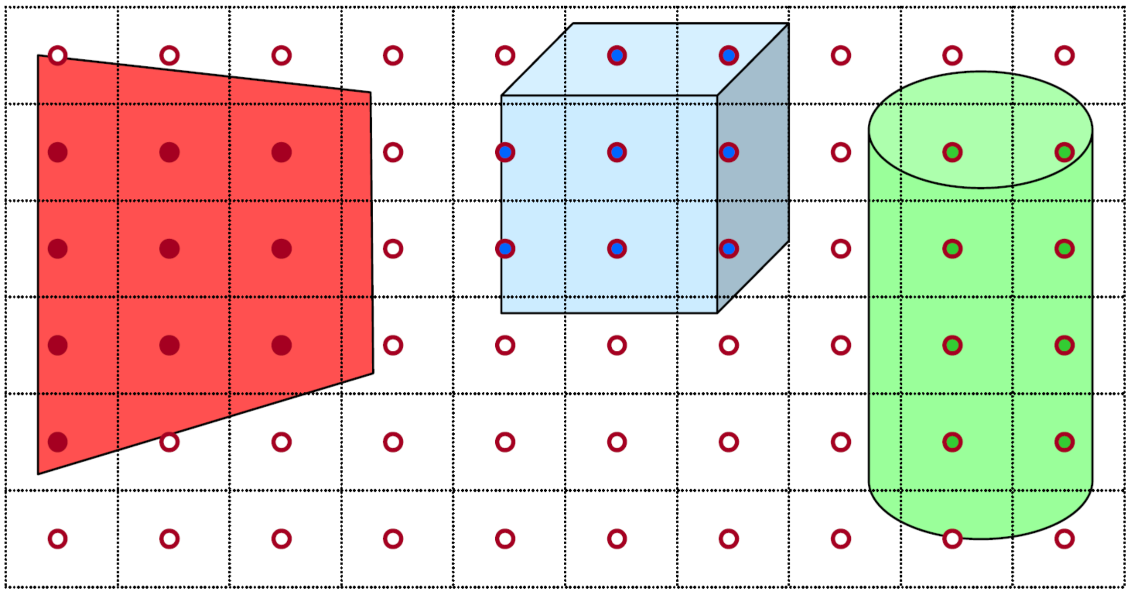
3D polygon rendering

- What steps are necessary to produce an image of a 3D scene?



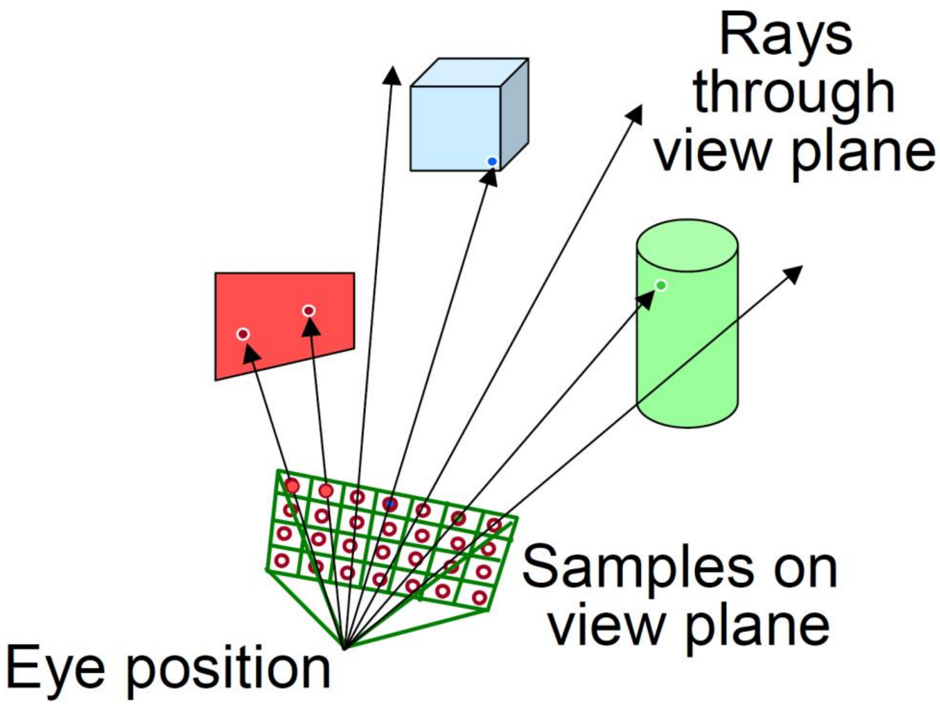
Ray Casting

- One approach is to cast rays from the camera…



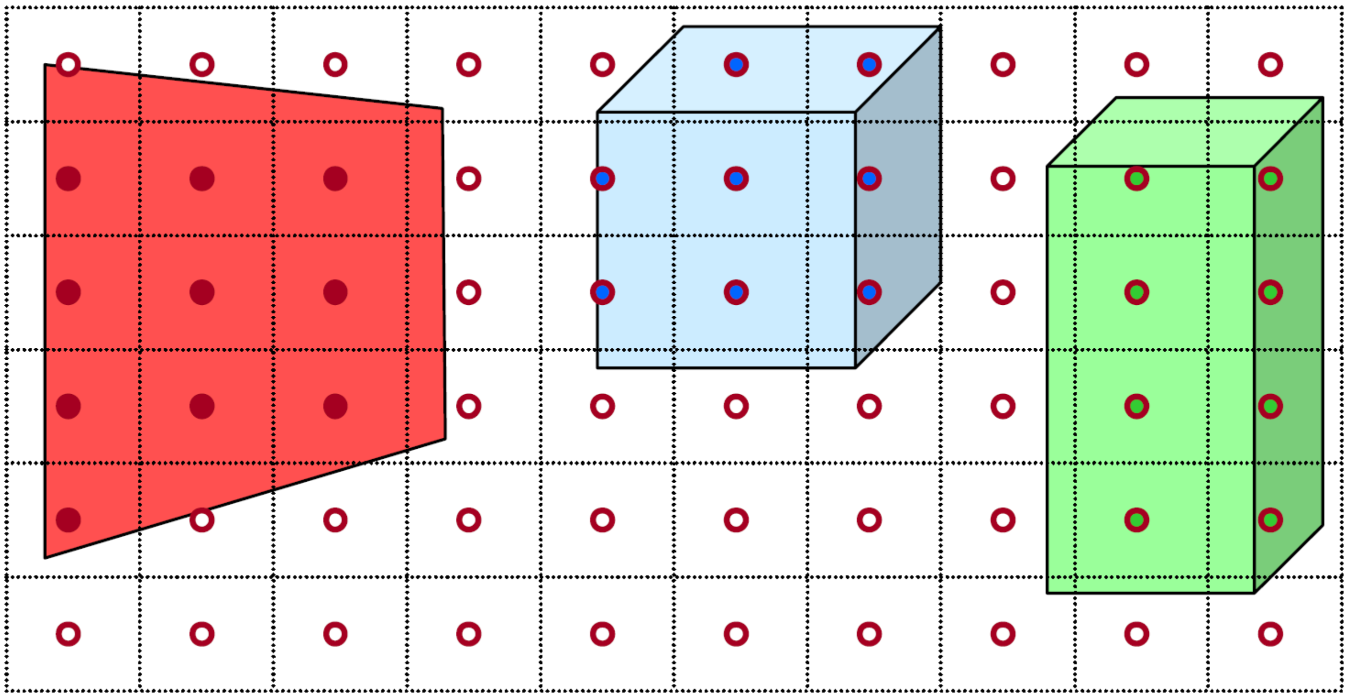
Ray Casting

- And find intersections with the scene…
- We are going to describe different approach this lesson



3D polygon rendering

- Second approach is called Rasterization
- Way how to efficiently draw primitives into screen space



How the lectures should look like #1

- Ask questions, please!!!
- Be communicative
- More active you are, the better for you!

Rasterization

3D rendering pipeline

3D polygons

Modeling Transformation

↓

Lighting

↓

Viewing Transformation

↓

Projection Transformation

↓

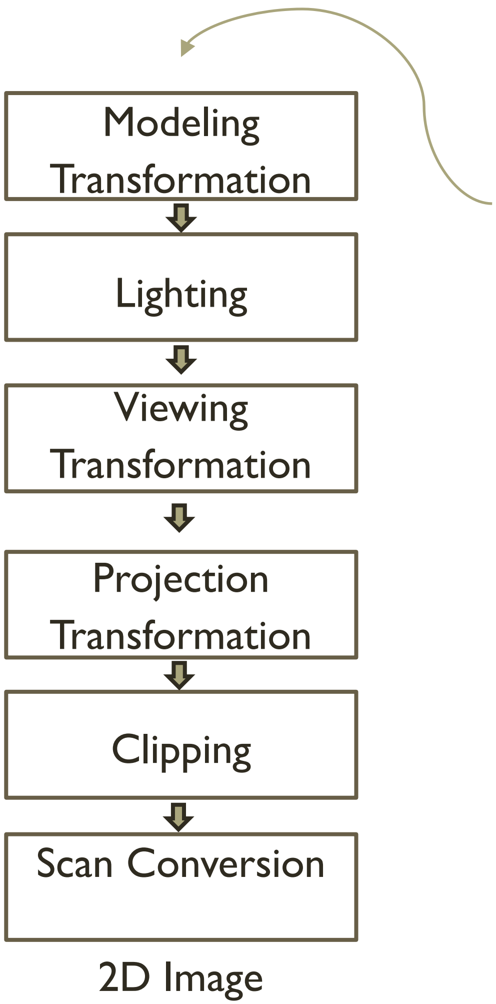
Clipping

↓

Scan Conversion

2D Image

3D rendering pipeline



array of vertex positions x,y,z { 0,1,0, 1,1,0, 1,0,0, 0,0,0}

OpenGL executes steps of the 3D rendering pipeline for each polygon

3D rendering pipeline

3D polygons

Modeling Transformation

↓

Lighting

↓

Viewing Transformation

↓

Projection Transformation

↓

Clipping

↓

Scan Conversion

2D Image

Transform into 3D world coordinate system

3D rendering pipeline

3D polygons

Modeling Transformation — Transform into 3D world coordinate system

↓

Lighting — Illuminate according to light

↓

Viewing Transformation

↓

Projection Transformation

↓

Clipping

↓

Scan Conversion

2D Image

3D rendering pipeline

3D polygons

Modeling Transformation — Transform into 3D world coordinate system

↓

Lighting — Illuminate according to light

↓

Viewing Transformation — Transform into 3D camera coordinate system

↓

Projection Transformation

↓

Clipping

↓

Scan Conversion

2D Image

3D rendering pipeline

3D polygons

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Illuminate according to light
Viewing Transformation	Transform into 3D camera coordinate system
Projection Transformation	Transform into 2D camera coordinate system
Clipping	
Scan Conversion	

2D Image

3D rendering pipeline

3D polygons

Modeling Transformation	Transform into 3D world coordinate system
⇩	
Lighting	Illuminate according to light
⇩	
Viewing Transformation	Transform into 3D camera coordinate system
⇩	
Projection Transformation	Transform into 2D camera coordinate system
⇩	
Clipping	Clip polygons outside of camera's view
⇩	
Scan Conversion	

2D Image

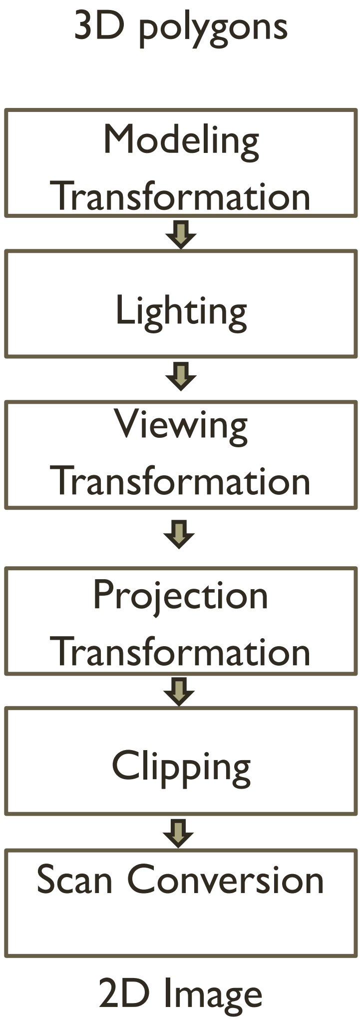
3D rendering pipeline

3D polygons

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Illuminate according to light
Viewing Transformation	Transform into 3D camera coordinate system
Projection Transformation	Transform into 2D camera coordinate system
Clipping	Clip polygons outside of camera’s view
Scan Conversion	Draw pixels

2D Image

3D rendering pipeline



- Model transformation
  - local → global coordinates
- View transformation
  - global → camera
- Projection transformation
  - camera → screen
- Clipping, rasterization, texturing & Lighting
  - might take place earlier

Transformations

3D polygons

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Illuminate according to light
Viewing Transformation	Transform into 3D camera coordinate system
Projection Transformation	Transform into 2D camera coordinate system
Clipping	Clip polygons outside of camera's view
Scan Conversion	Draw pixels

2D Image

Transformations

P(x, y, z)

⇩ 3D Object coordinates

Modeling Transformation

⇩ 3D World coordinates

Viewing Transformation

⇩ 3D Camera coordinates

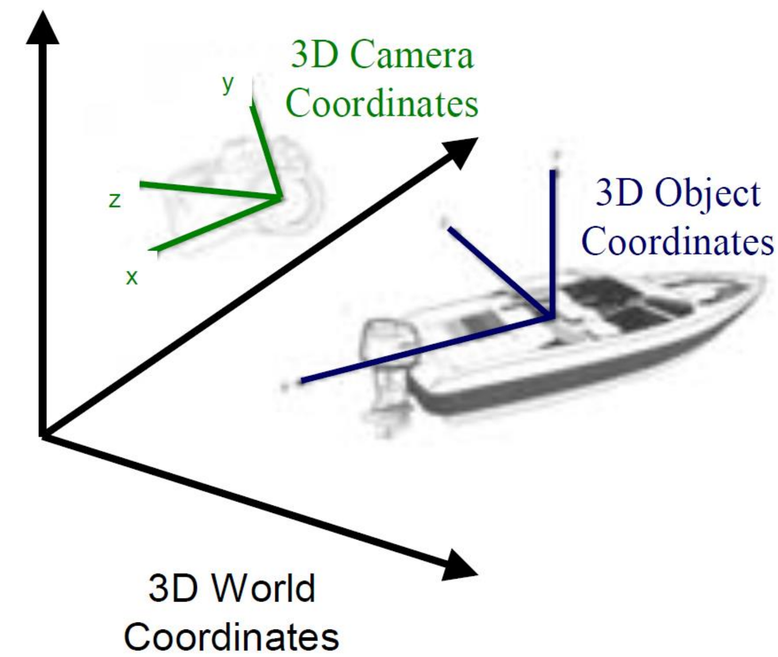
Projection Transformation

⇩ 2D Camera coordinates

Window to Viewport Transformation

⇩ 2D Image coordinates

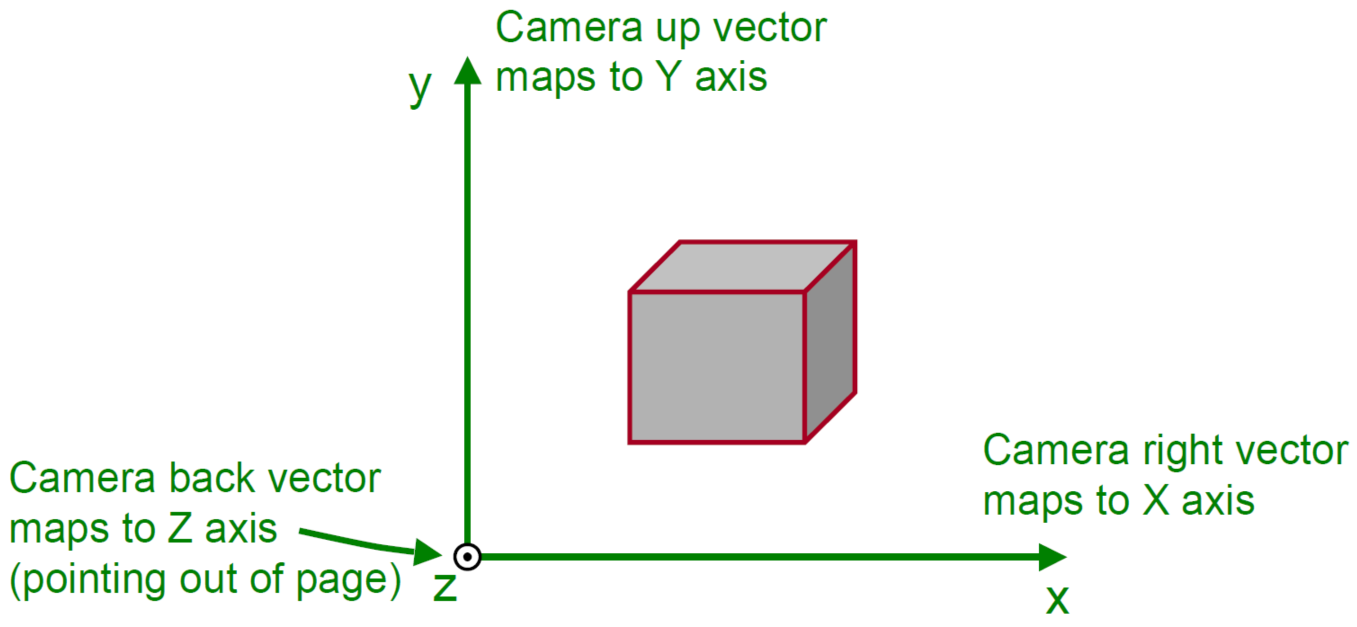
P'(x', y')



Transformations map points from one coordinate system to another

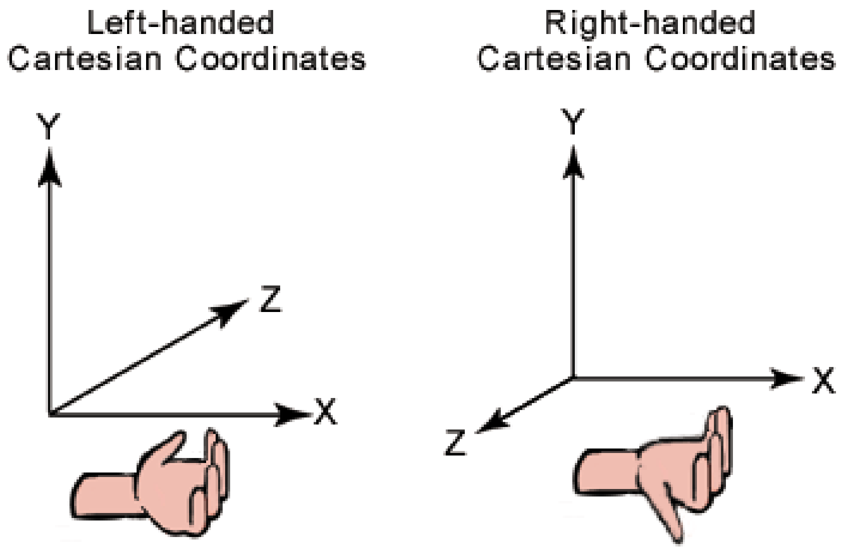
Camera coordinates

- Canonical coordinate system
  - Convention is right-handed (looking down -z)
  - Convenient for projection, clipping etc.



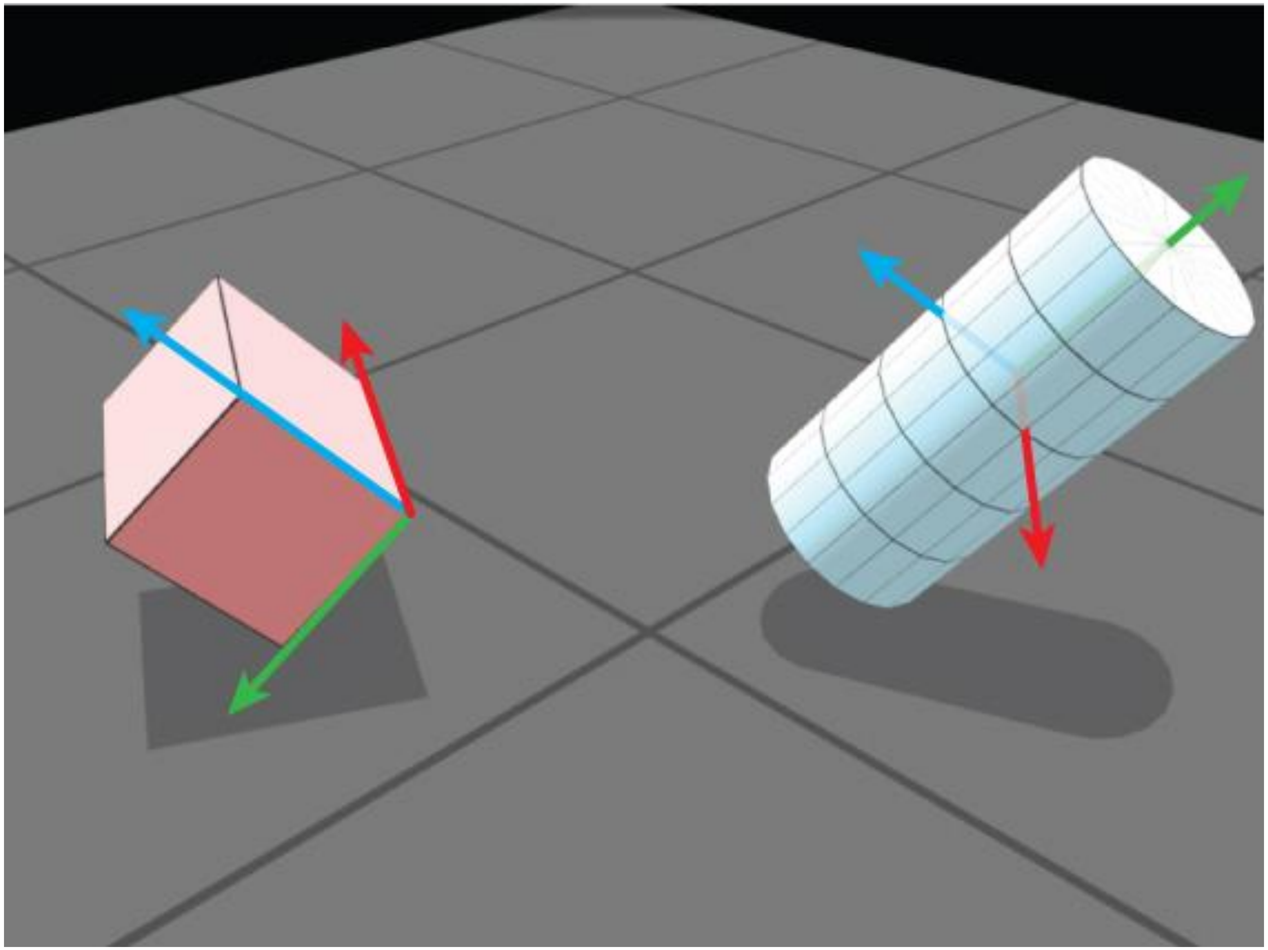
Coordinate systems

- DirectX <= 9, left handed only



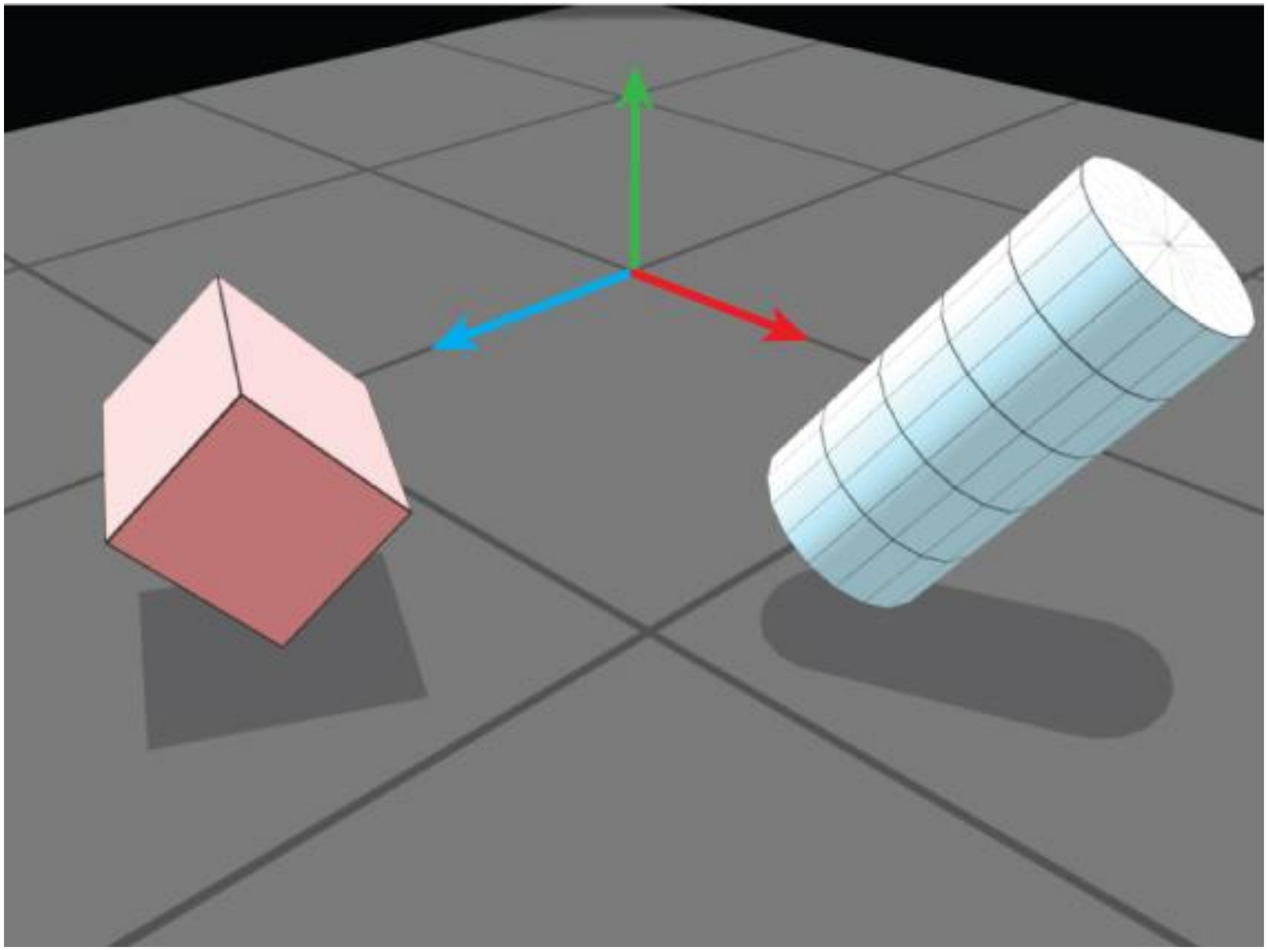
Local coordinates

- Each object has its own coordinate system



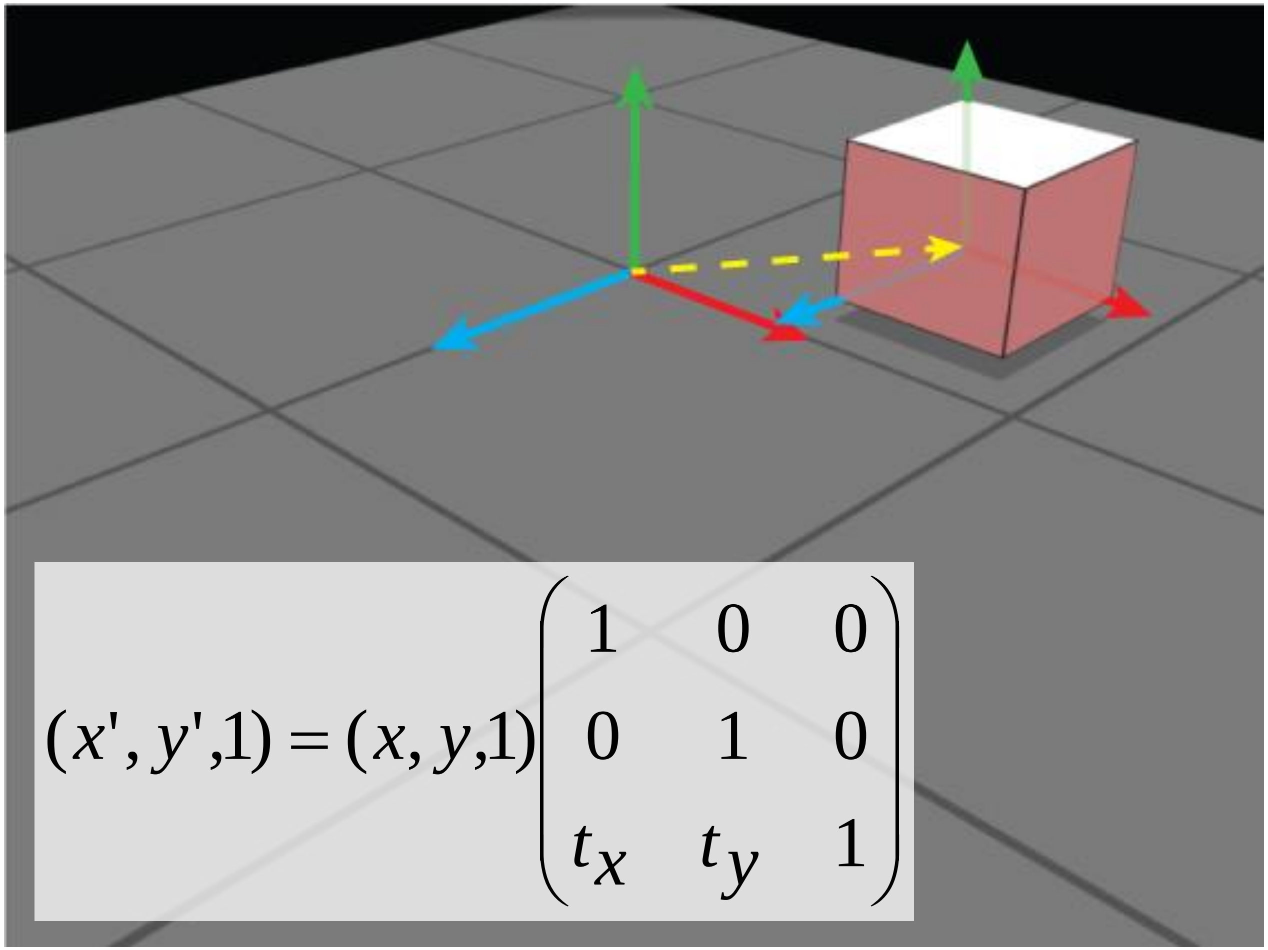
Global coordinates

- One system for the whole scene



Local → Global coordinates

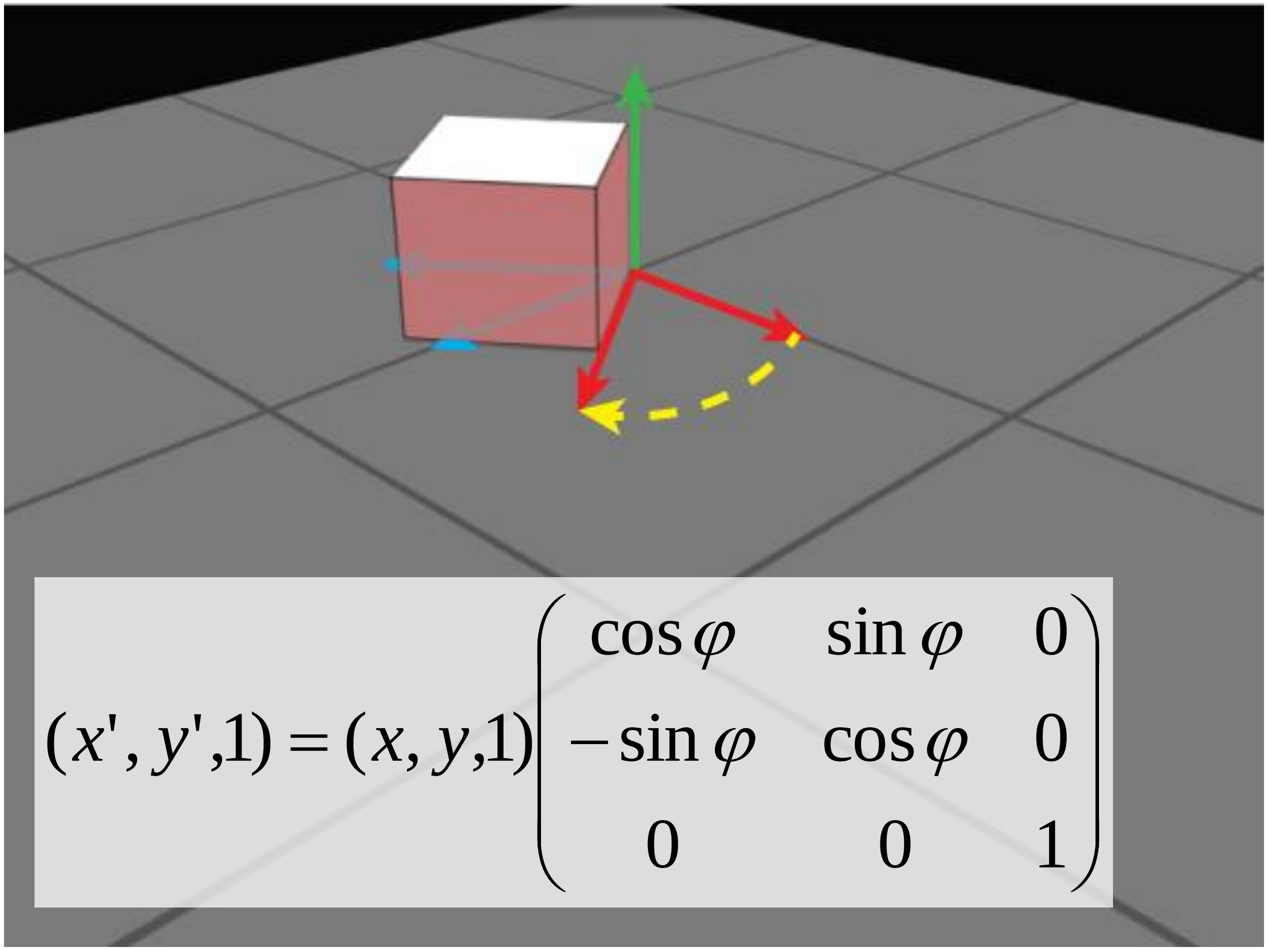
- Translation



$$(x', y', 1) = (x, y, 1)\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

Local → Global coordinates

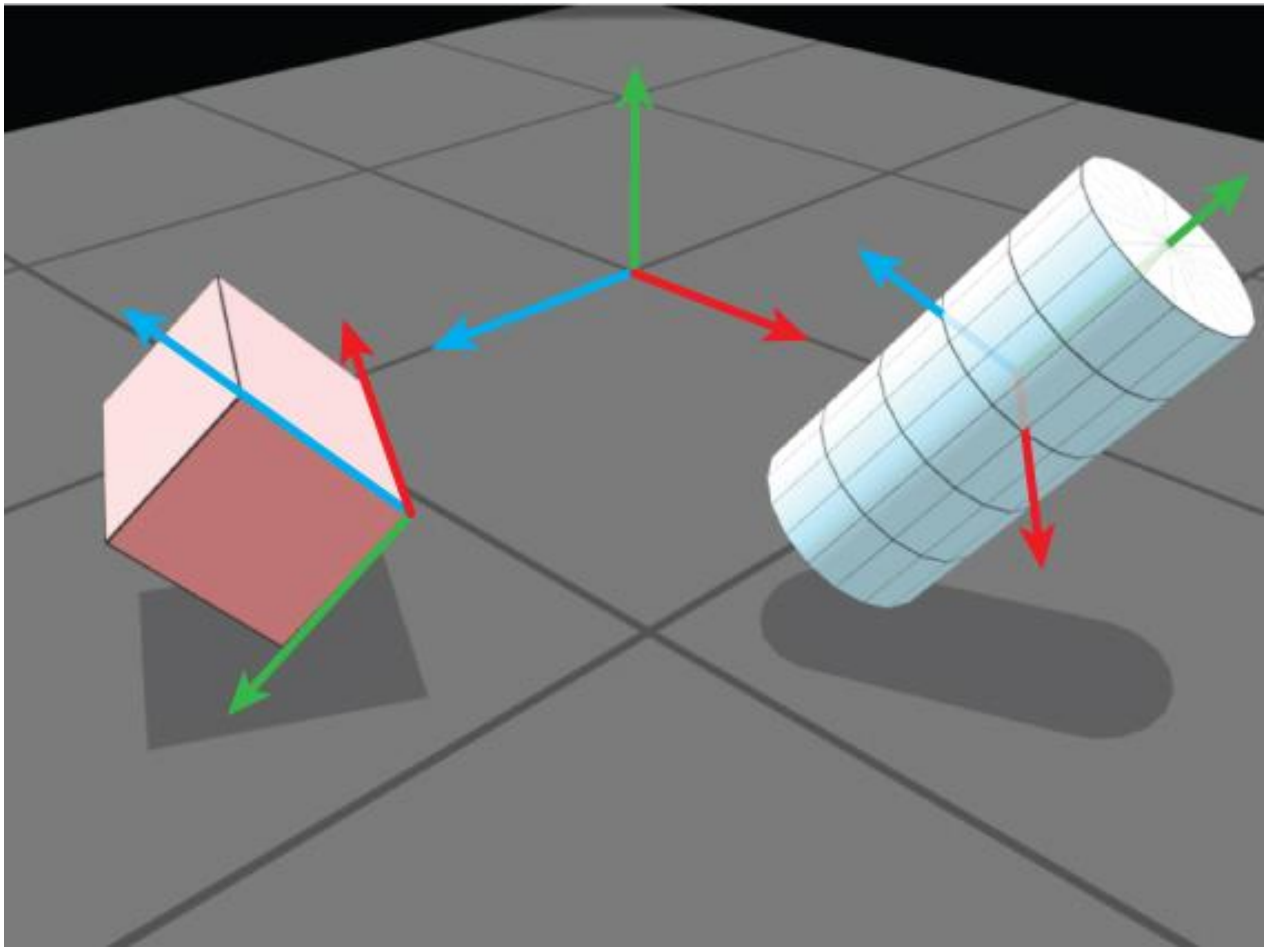
- Rotation



$$(x', y', 1) = (x, y, 1)\begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

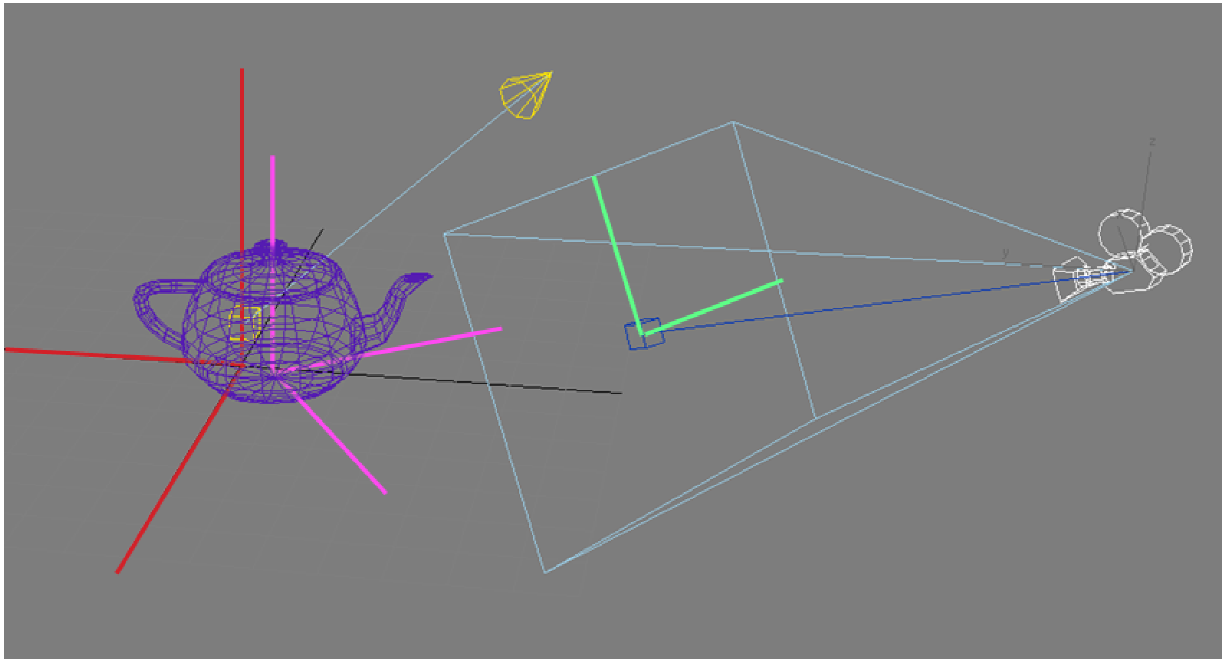
Local → Global coordinates

- All transformations combined



Transformations

- Transformation from one coordinate system to another one is a composition of partial transformations:
  - Translation
  - Rotation
  - Scaling

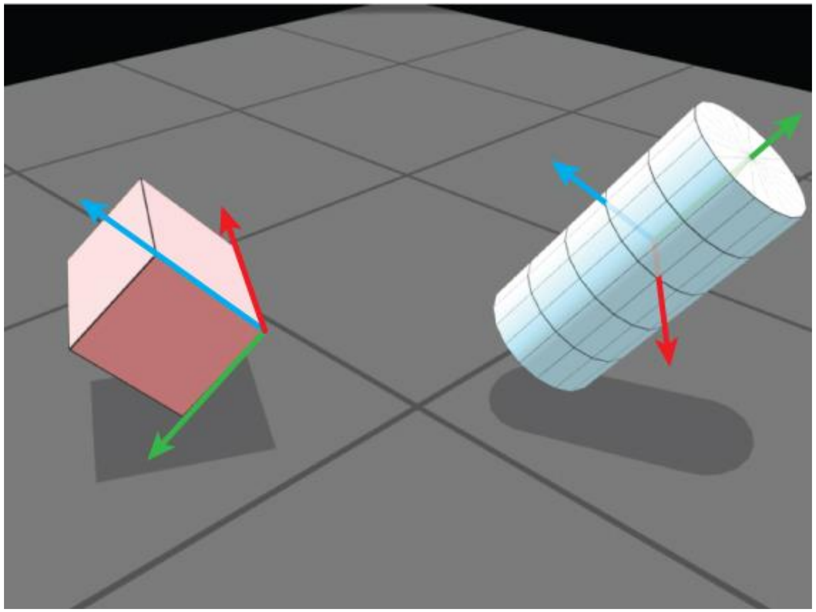


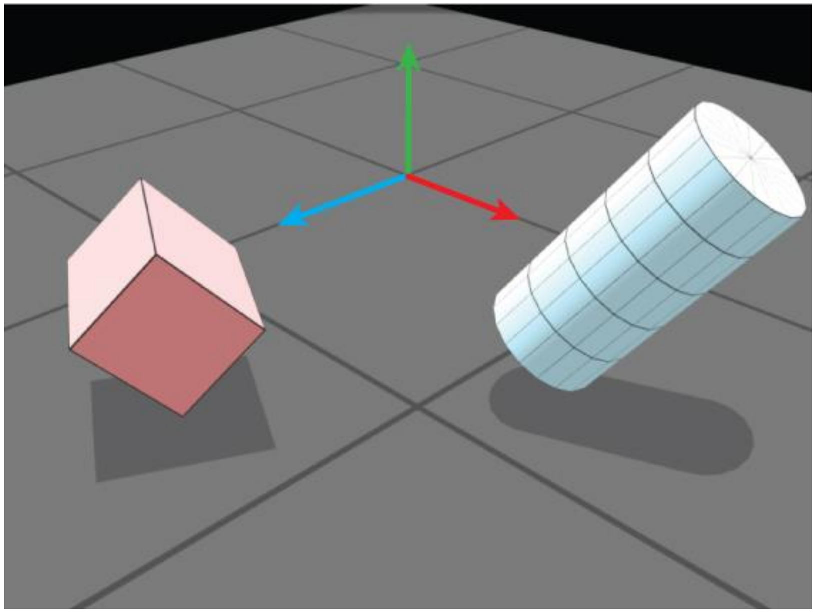
All transformations

- Model transformation
  - Unify coordinates by transforming local to global coordinates
- View transformation
  - Transform global coordinates so that they are aligned with camera coordinates
  - To make projection computable

Model transformation

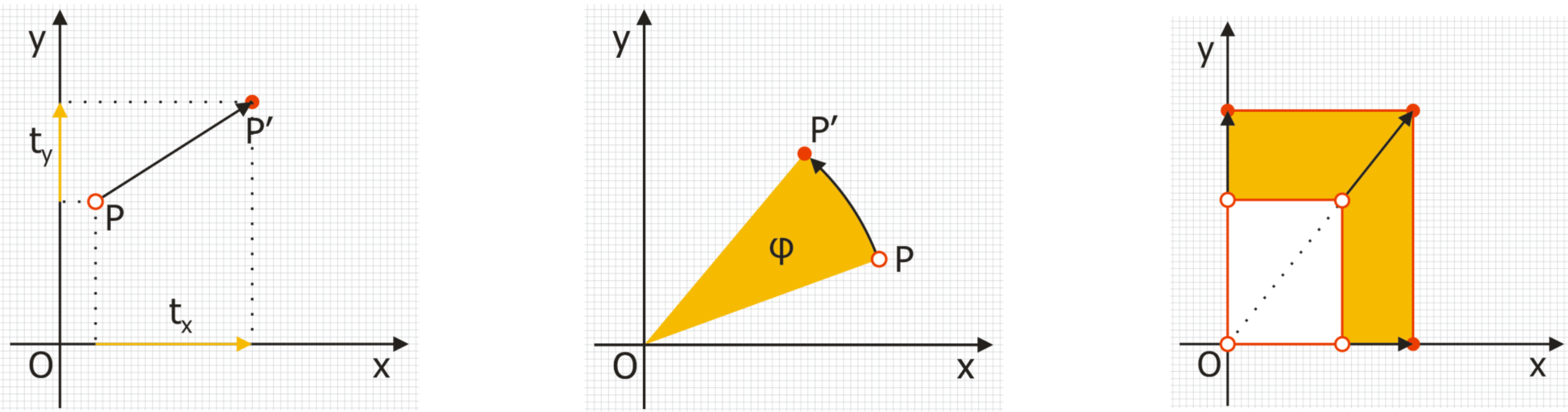
- Transformation local → global
- Combination of rotate, translate, scale
- Matrix multiplication





Model transformation

- Translation, rotation, scaling



$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{pmatrix} \quad \begin{pmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

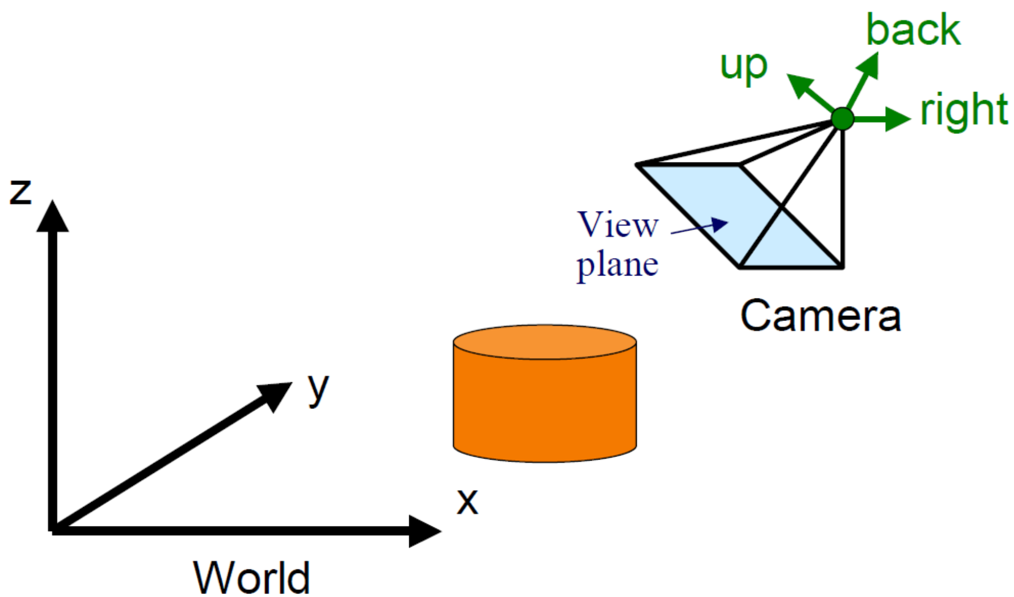
Global→camera coordinates

- $T * R_y * R_x$
  - Translation, rotation, rotation
- $T * R_y * R_x * R_z$
  - if the camera is rolled

- Projection P
  - orthogonal, perspective, isometric ...

Viewing Transformation

- Mapping from world to camera coordinates
  - Eye position maps to origin
  - Right vector maps to X axis
  - Up vector maps to Y axis
  - Back vector maps to Z axis



Finding the Viewing Transformation

- We have the camera (in world coordinates)
- We want *T* taking objects from world to camera

$$p^C = Tp^W$$

- Trick: find T taking objects in camera to world

$$p^W = T^{-1}p^C$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Finding the Viewing Transformation

- Trick: Map from camera coordinates to world
  - Origin maps to eye position
  - z axis maps to Back vector
  - y axis maps to Up vector
  - x axis maps to Right vector

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} r_x & u_x & b_x & e_x \\ r_y & u_y & b_y & e_y \\ r_z & u_z & b_z & e_z \\ r_w & u_w & b_w & e_w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

  - To get T^{-1} we just need to invert T

Finding the Viewing Transformation

- Trick: Map from camera coordinates to world
  - Origin maps to eye position
  - z axis maps to Back vector
  - y axis maps to Up vector
  - x axis maps to Right vector

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} r_x & u_x & b_x & e_x \\ r_y & u_y & b_y & e_y \\ r_z & u_z & b_z & e_z \\ r_w & u_w & b_w & e_w \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

  - To get T^{-1} we just need to invert T

Vectors vs Positions

- There is a fundamental difference between vectors and positions in homogeneous coordinates!
- Position
  - In homogeneous coordinates p = {x, y, z, 1}
  - Can be **moved** so translation will apply
- Vector
  - In homogenous coordinates v = {x, y, z, 0}
  - Cannot be moved, its just **direction**

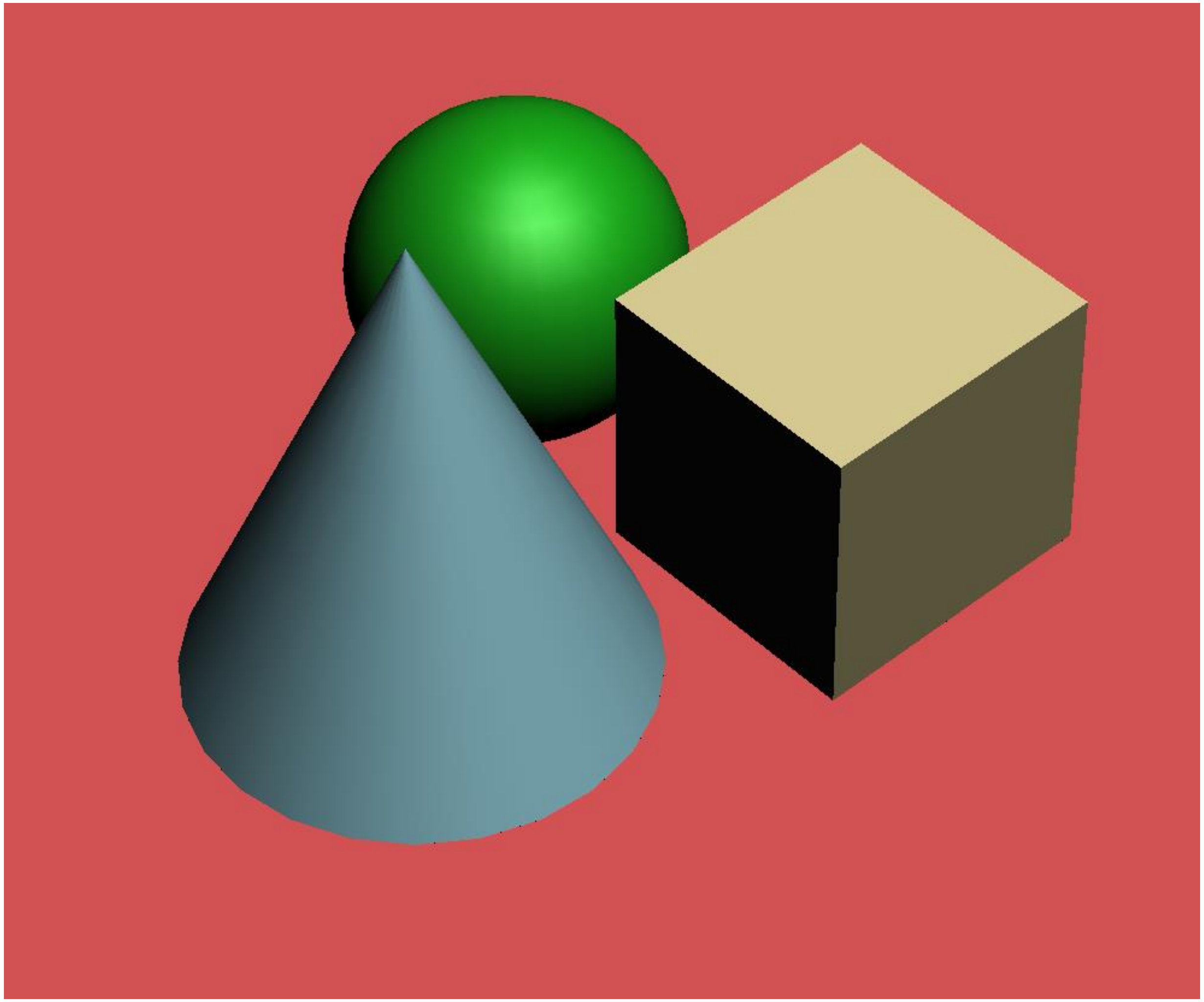
Projections summary

Projection types

- Orthogonal

Projection types

- Parallel

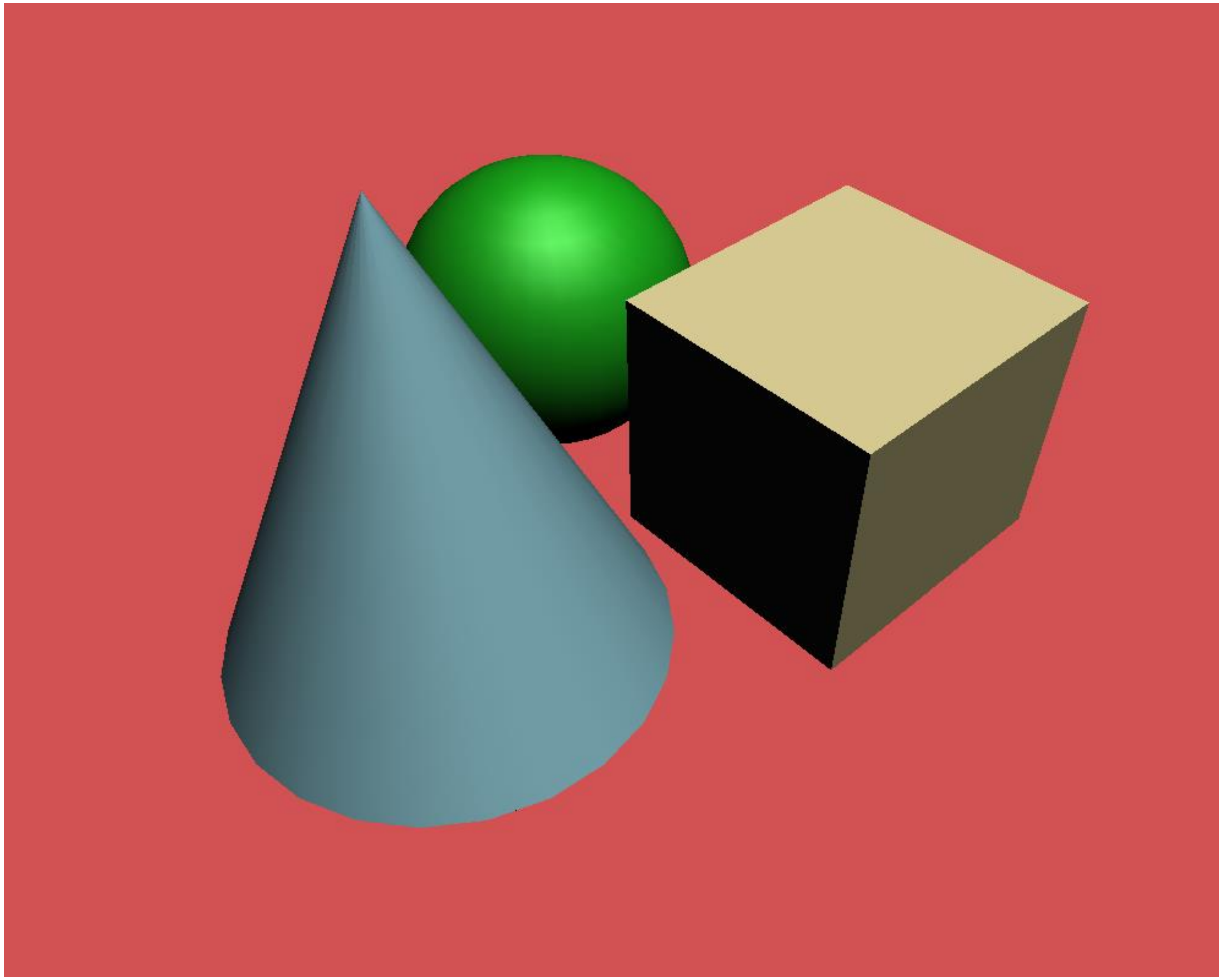


Projection types

- Isometric (parallel but not orthogonal)

Projection types

- Perspective



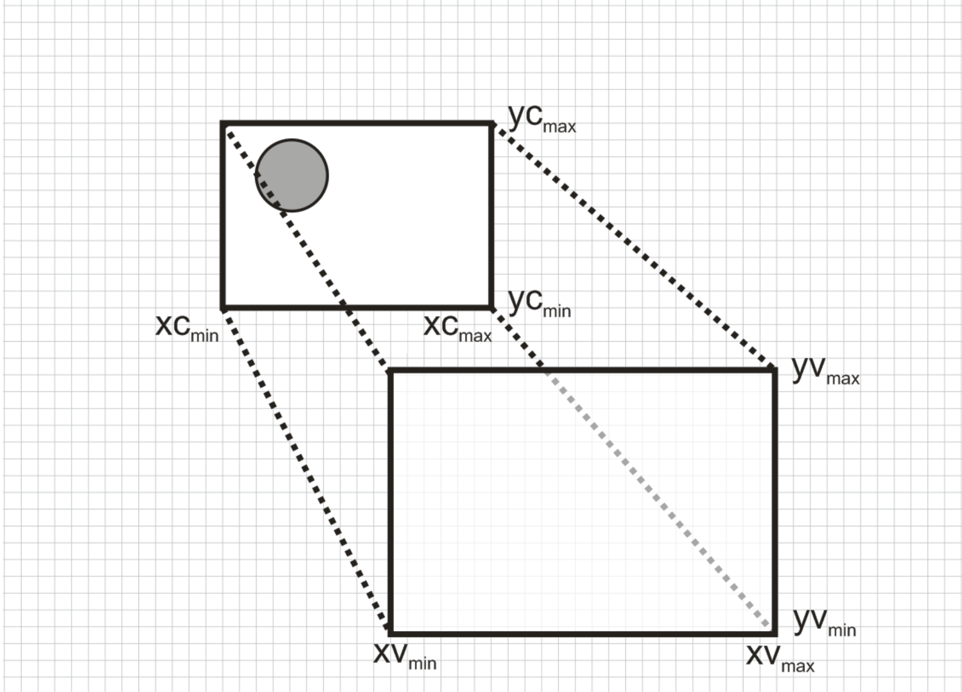
Projection types

- Perspective

Viewport transformation

Viewport transformation

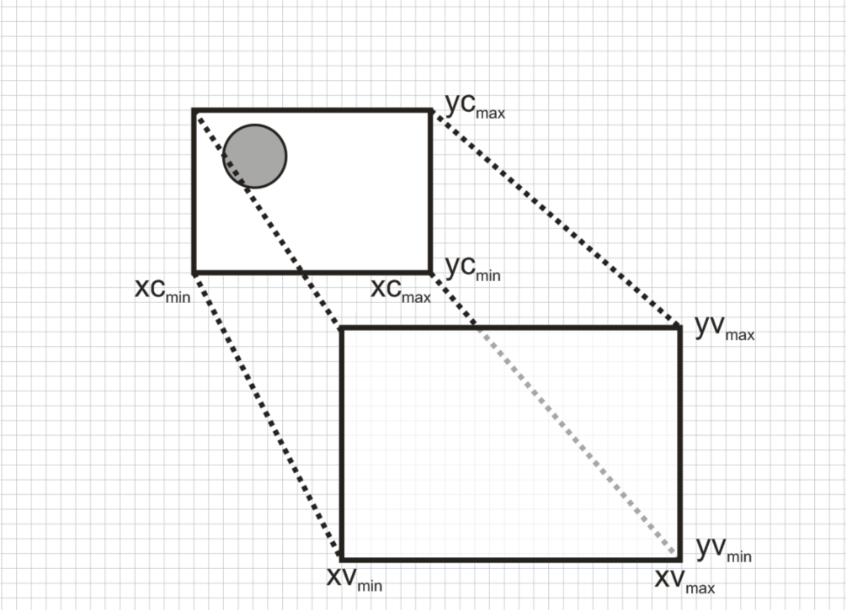
- Global coordinates
  - e.g. (-50..50 cm, -50..50 cm, -50..50 cm)
- Camera coordinates
  - e.g. (-1..1, -1..1, -1..1)
- Viewport (window)
  - e.g. (0..1200 px, 0..800 px)



Viewport transformation

$$s_x = \frac{xv_{\max} - xv_{\min}}{xc_{\max} - xc_{\min}}$$

$$s_y = \frac{yv_{\max} - yv_{\min}}{yc_{\max} - yc_{\min}}$$



$$(x_v, y_v, 1) = (x_p, y_p, 1)\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ -s_x xc_{\min} + xv_{\min} & -s_y yc_{\min} + yv_{\min} & 1 \end{pmatrix}$$

3D rendering pipeline

3D polygons

Modeling Transformation	Transform into 3D world coordinate system
⇩ Lighting	Illuminate according to light
⇩ Viewing Transformation	Transform into 3D camera coordinate system
⇩ Projection Transformation	Transform into 2D camera coordinate system
⇩ Clipping	Clip polygons outside of camera’s view
⇩ Scan Conversion	Draw pixels

2D Image

3D rendering pipeline

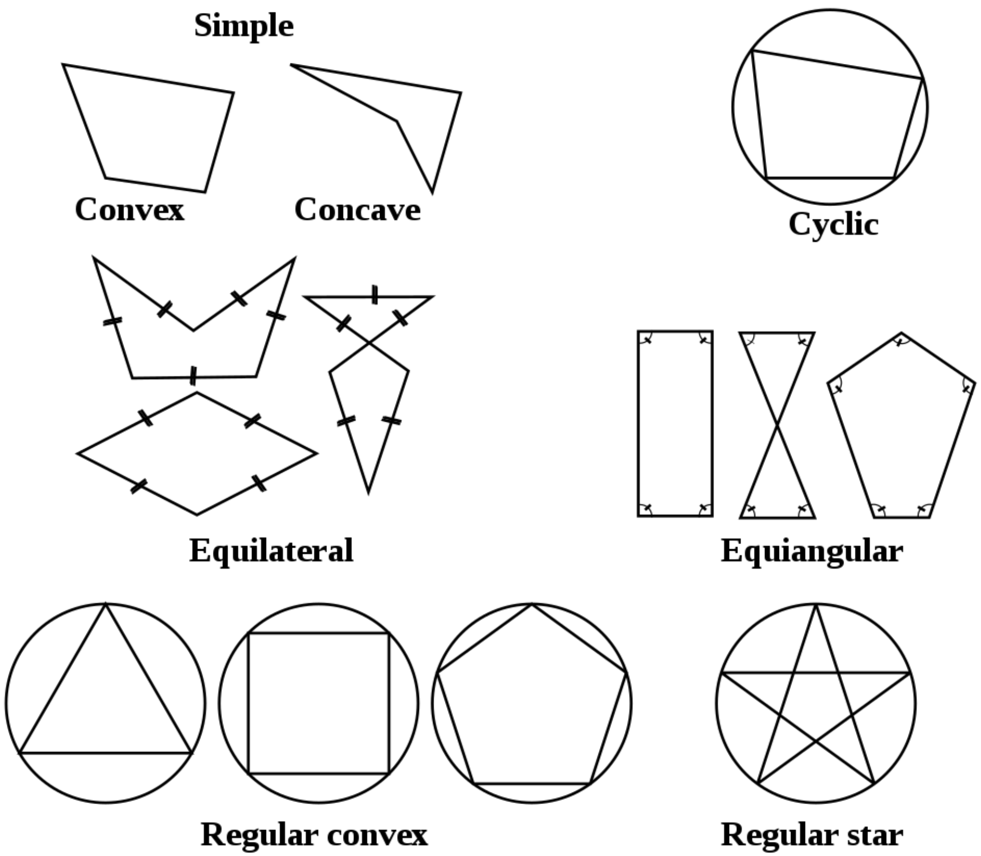
3D polygons

Modeling Transformation	Transform into 3D world coordinate system
Lighting	Illuminate according to light
Viewing Transformation	Transform into 3D camera coordinate system
Projection Transformation	Transform into 2D camera coordinate system
Clipping	Clip polygons outside of camera's view
Scan Conversion	Draw pixels

2D Image

3D polygon rendering

- Closed sequence of lines



Line rasterization

Digital Differential Analyzer

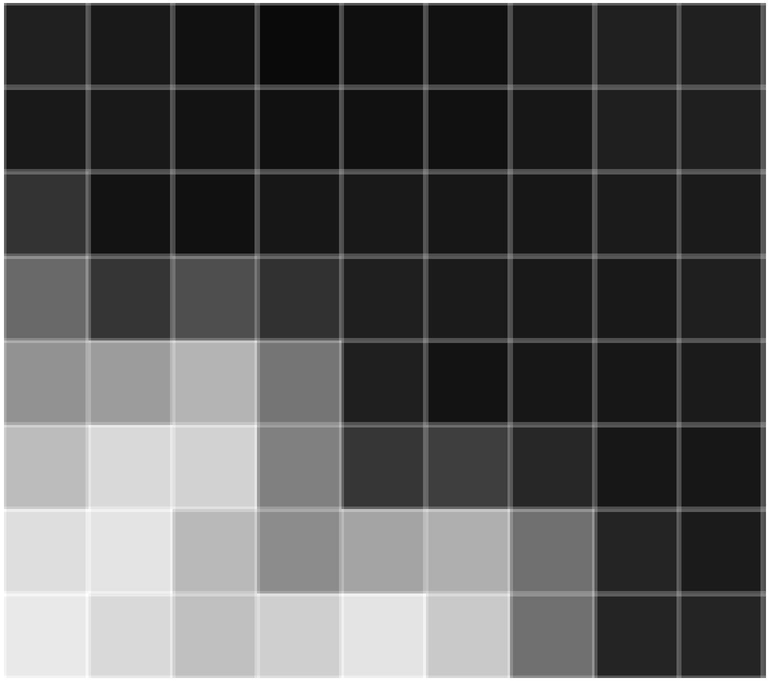
or

Bresenham algorithm

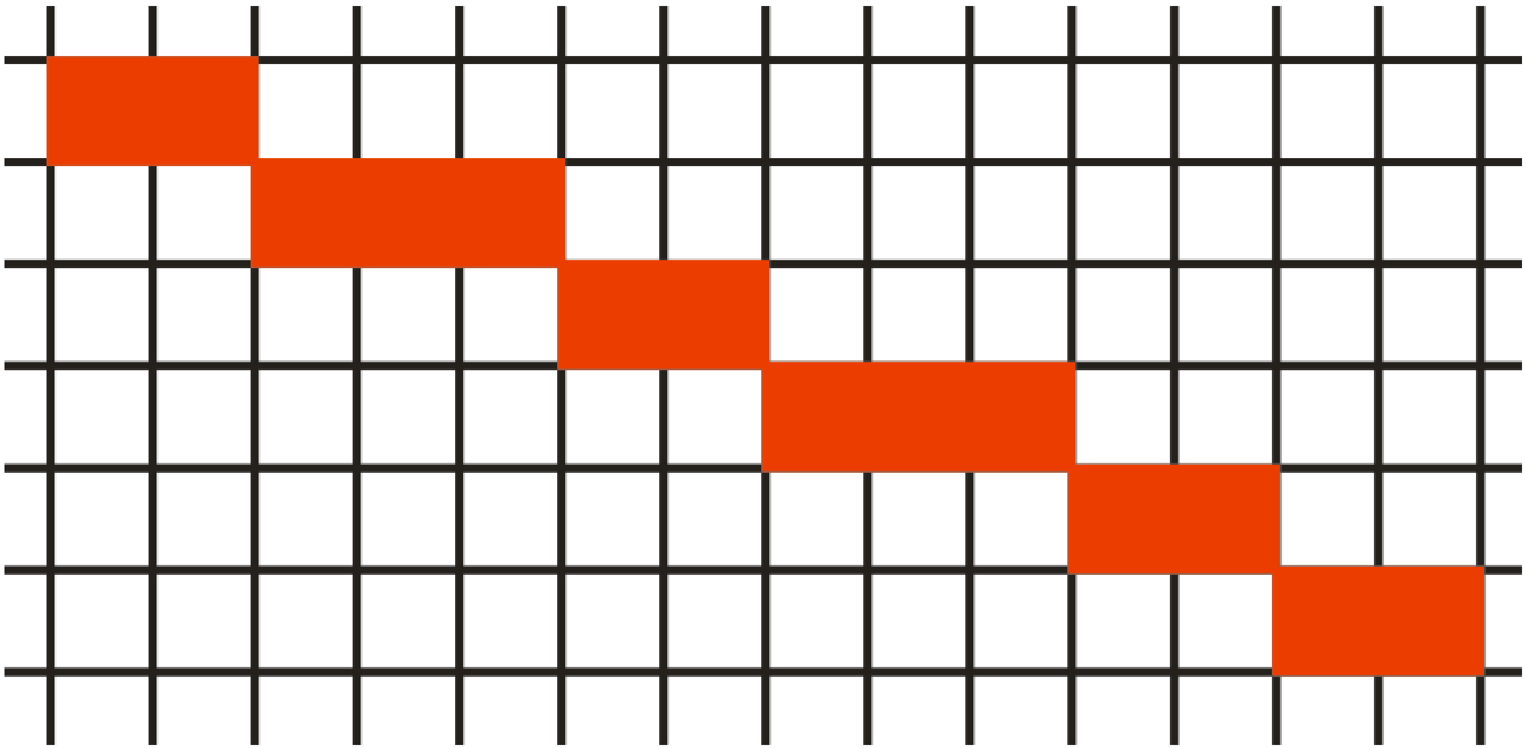
General problem

- Given a continuous geometric representation of an object

- Decide which pixels are occupied by the object

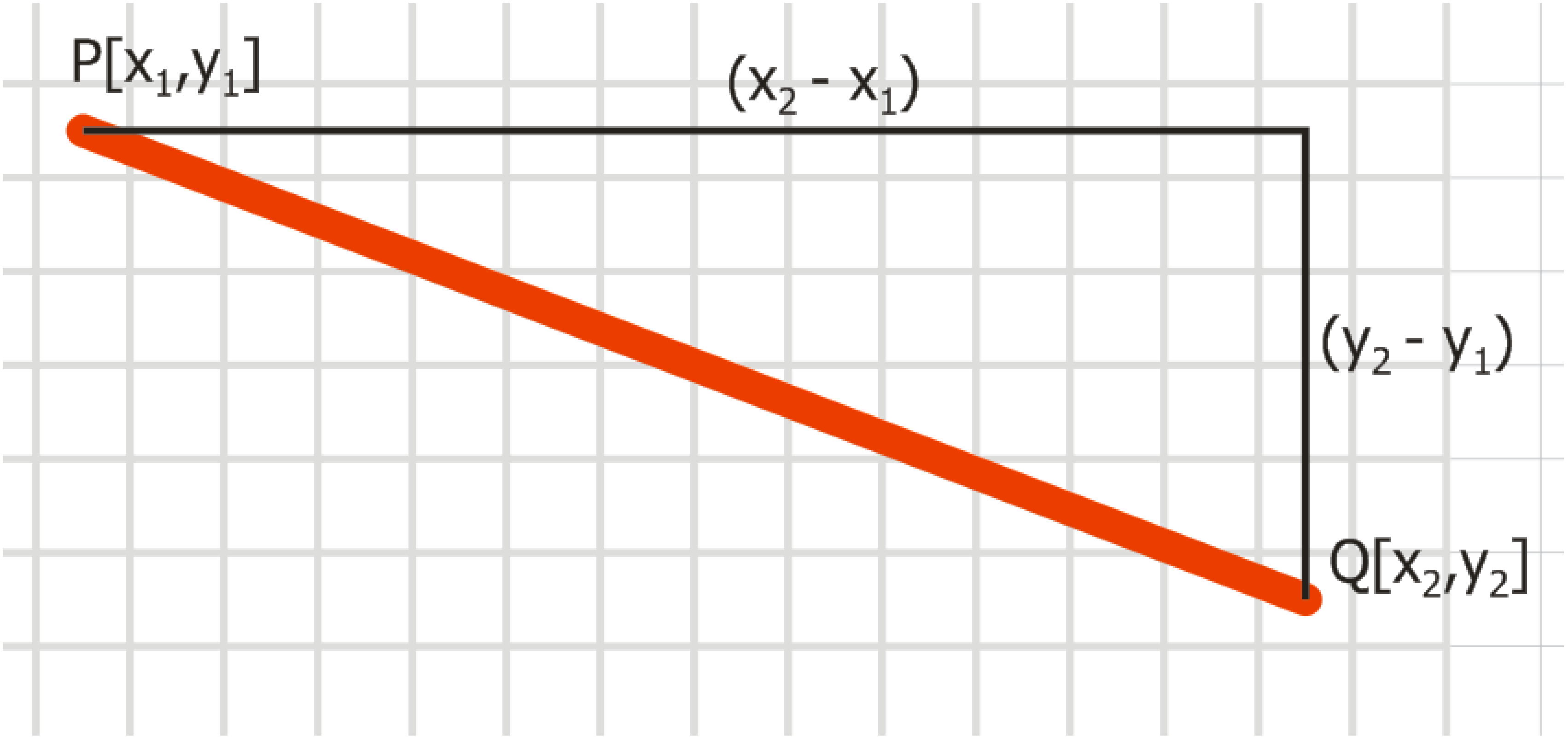


General problem



Digital Differential Analyzer

$dd = (y_2 - y_1) / (x_2 - x_1)$: float

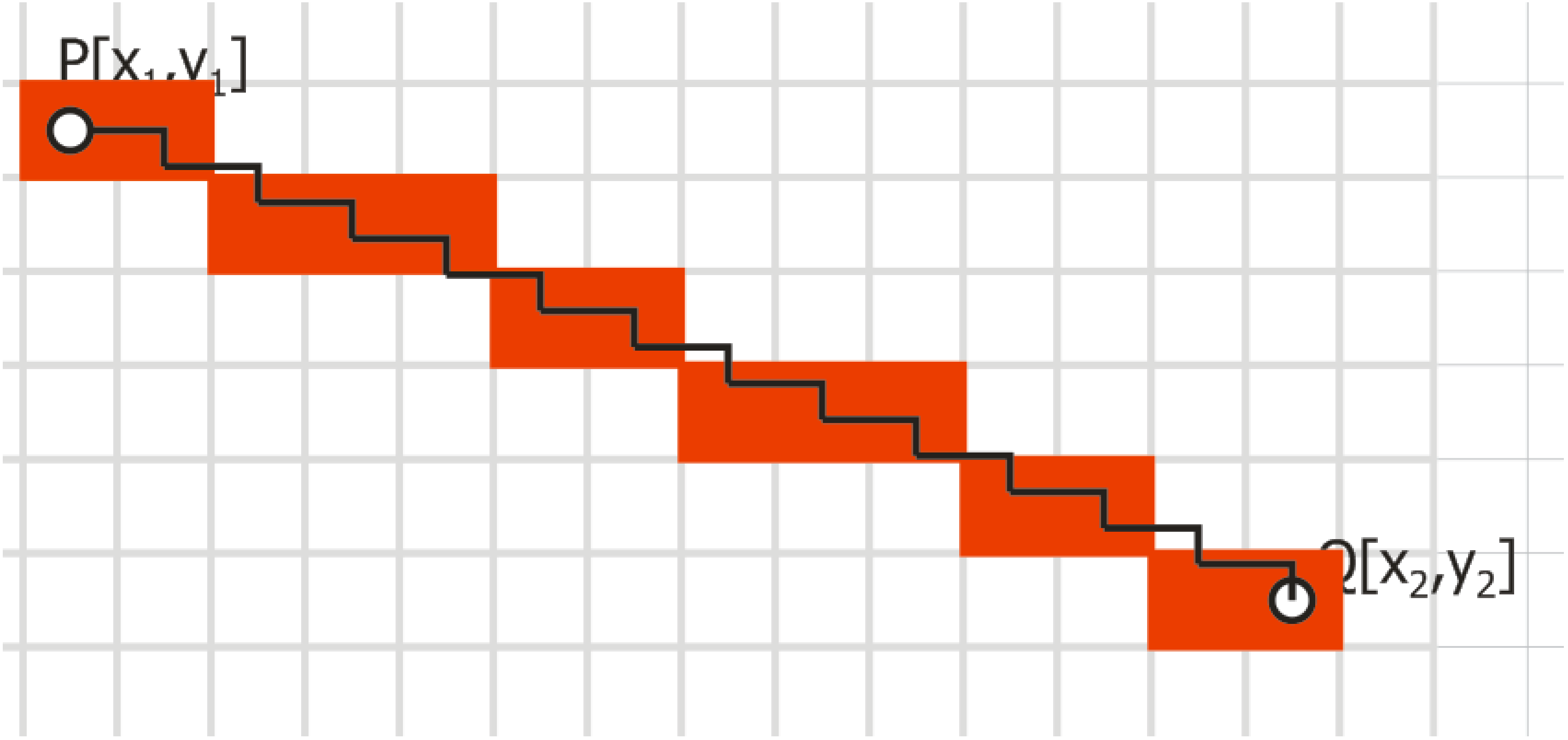


Digital Differential Analyzer

Pseudocode:

```
y = y1
for x = x1 to x2
   begin
     setpixel (x, round(y))
     y = y + dd
   end
```

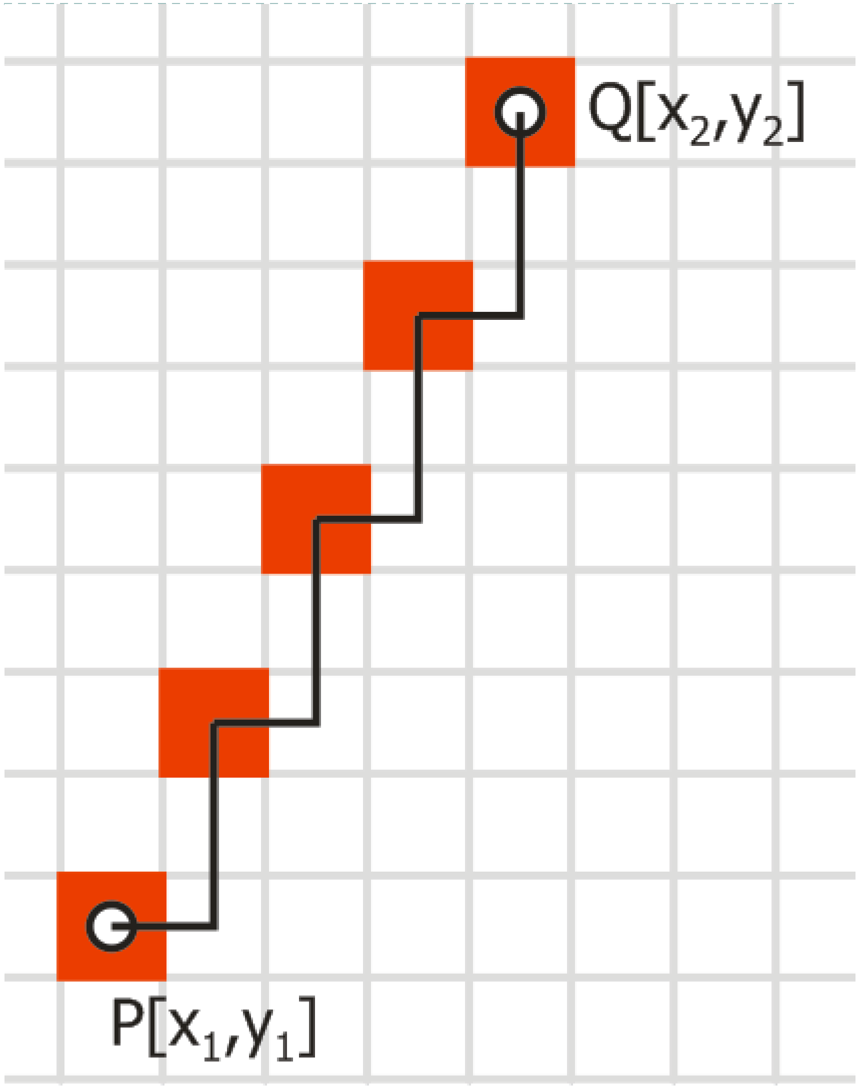
Digital Differential Analyzer



Watch for line slope

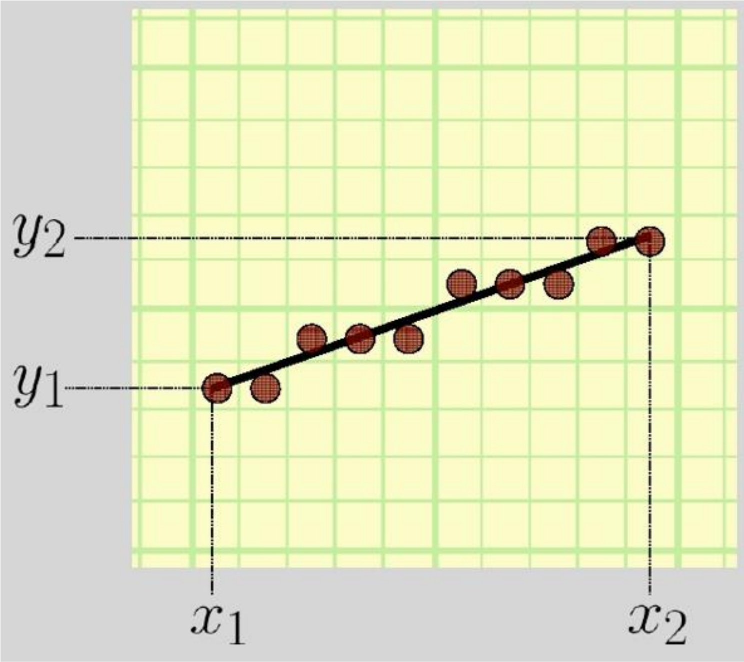
if abs(dd) > 1

exchange x↔y
in algorithm



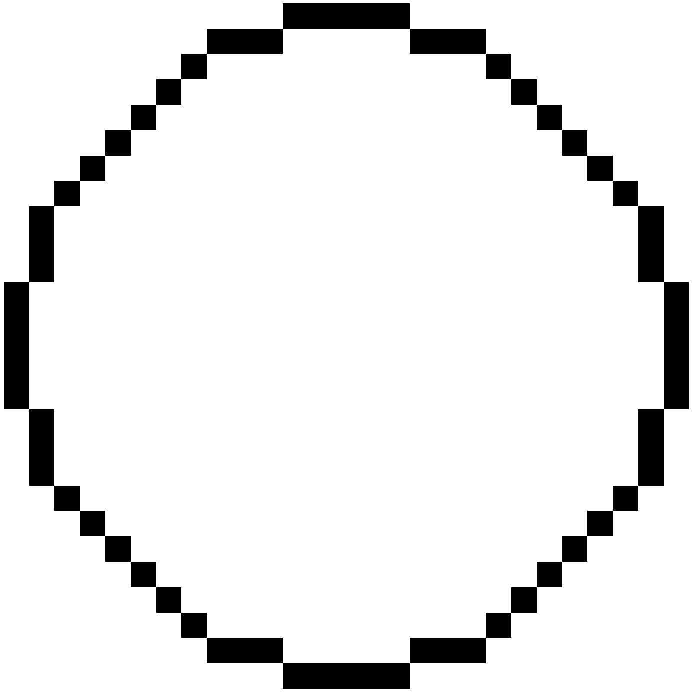
Bresenham algorithm

- DDA requires floating point
- Bresenham works with integers only
- main idea: for each x there are only 2 possible y values, pick the one with the smaller error. accumulate error over iterations.
- modify for other slopes and orientations



Circle, ellipse rasterization

- Bresenham for circles (midpoint algorithm)
- Can be modified for ellipses



Filled polygon rasterization

Scanline algorithm

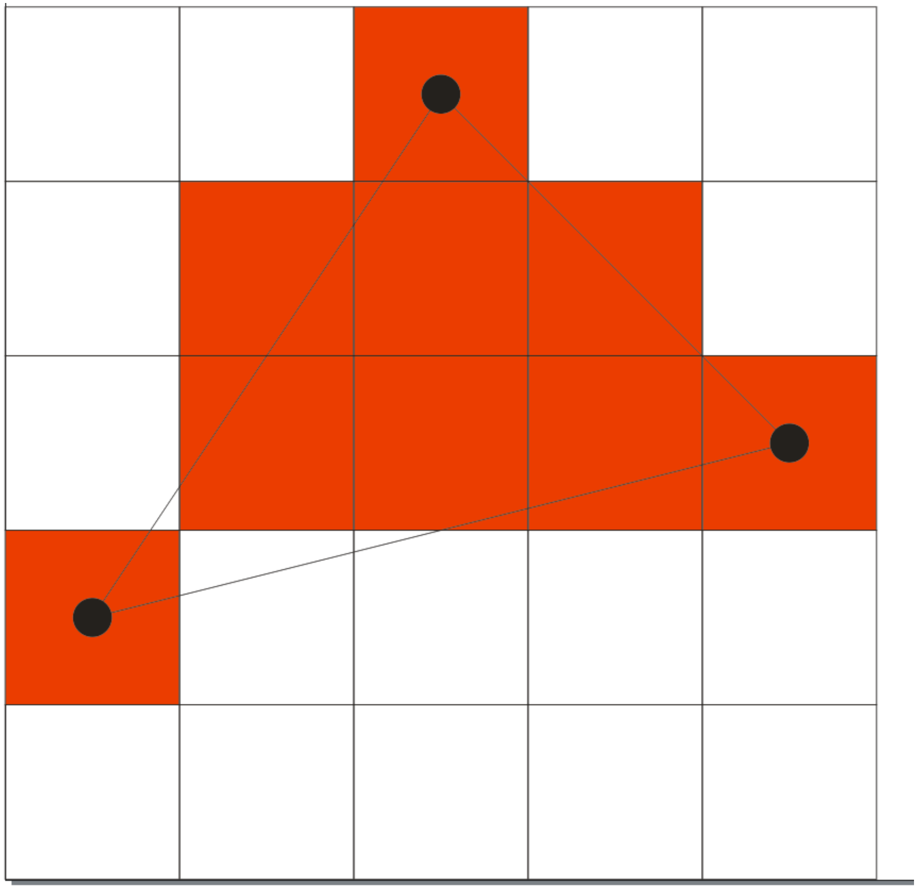
Polygon rasterization

Scanline algorithm:

For each scan line:

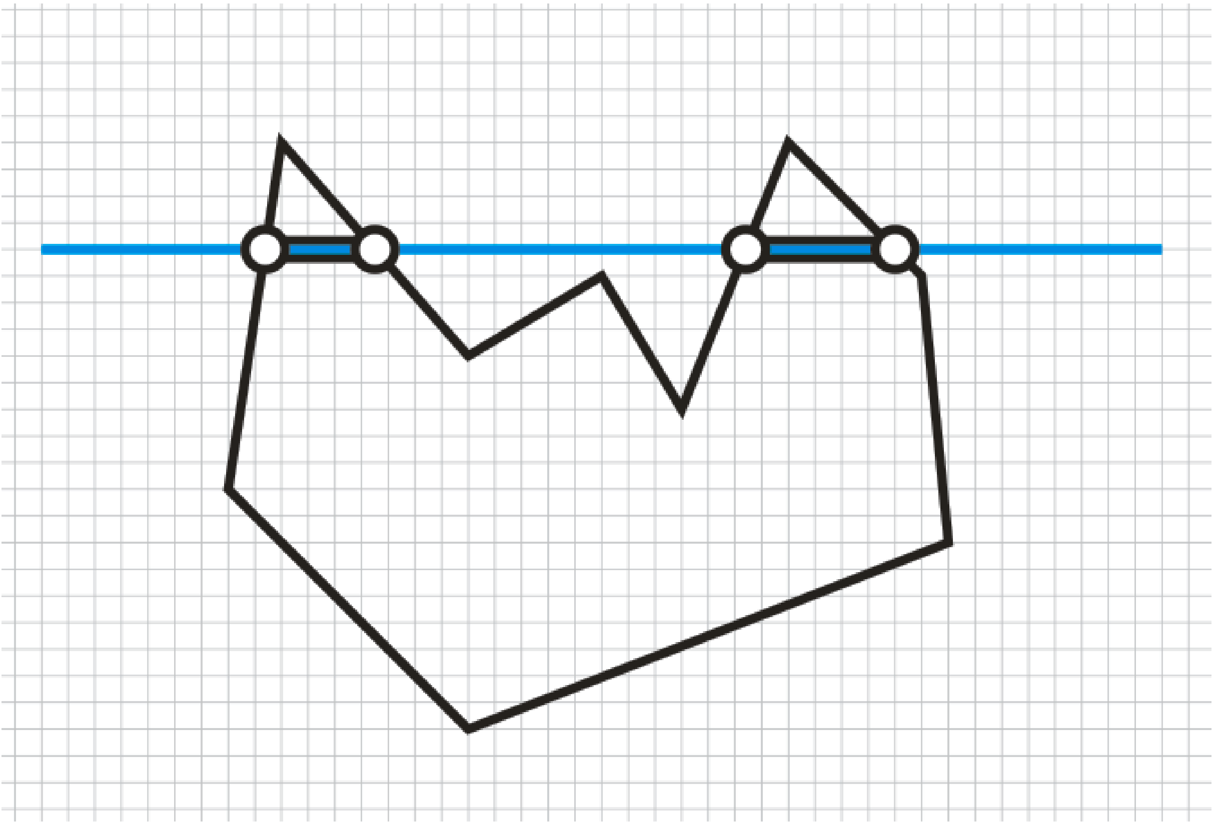
1. Find the intersections of polygon and the scan line
2. Sort the intersections by x coordinate
3. Fill the pixels between subsequent pairs of intersections

Scan-line algorithm



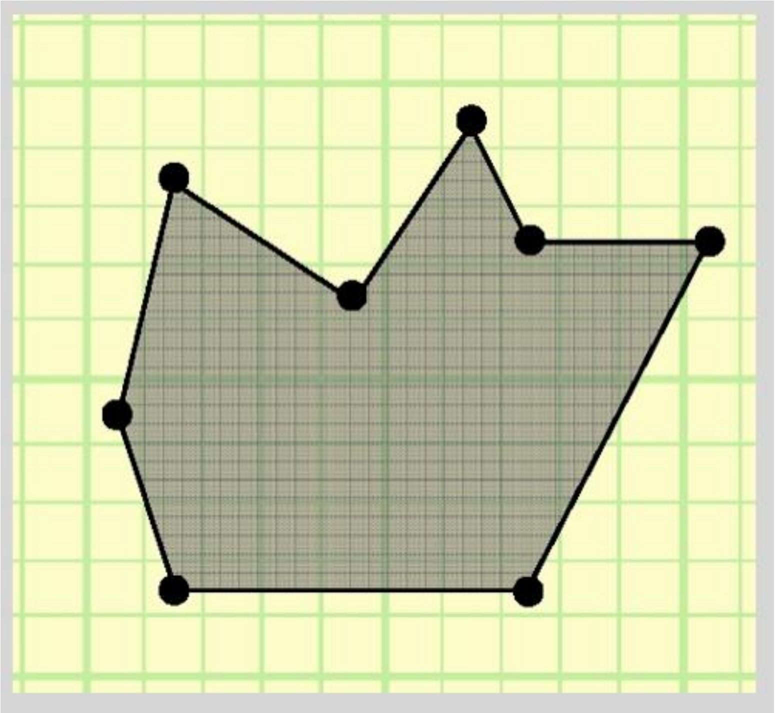
Scan-line algorithm

- (works also for non-convex polygons)



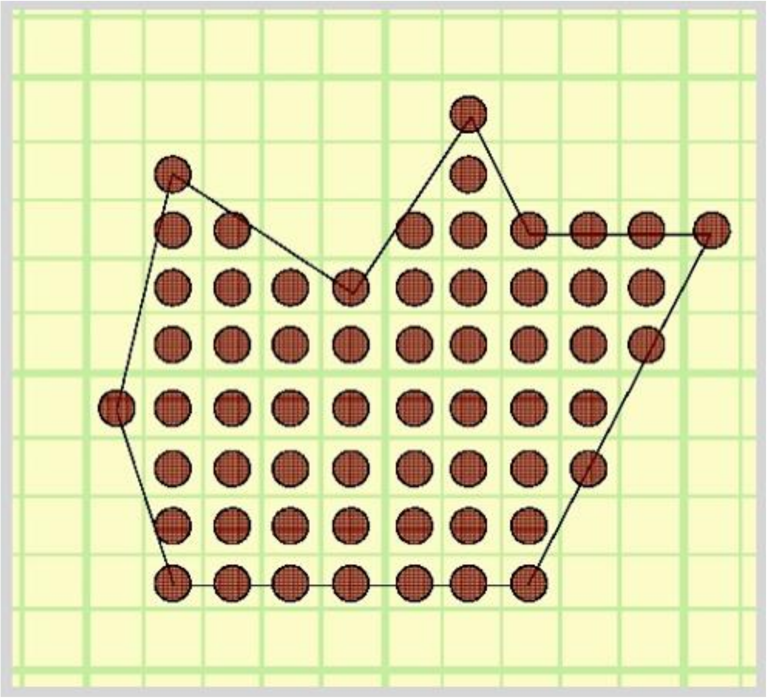
Filled polygon

- How to draw all pixels inside a polygon?



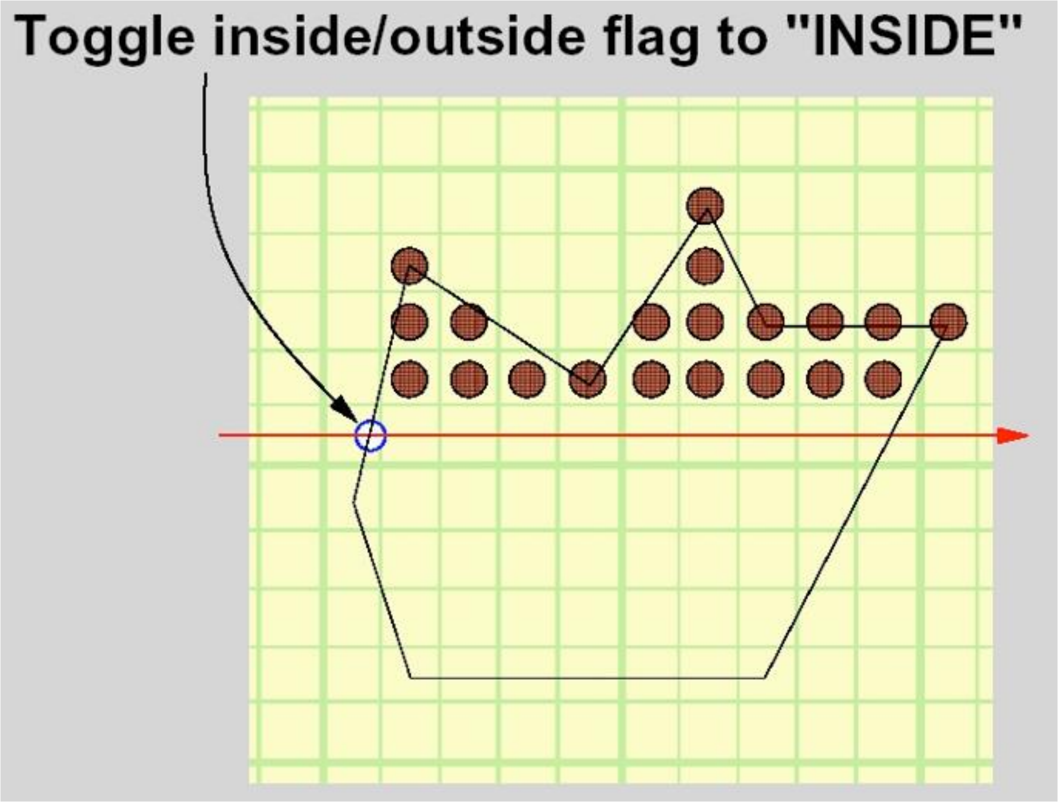
Filled polygon

- We need to determine INSIDE / OUTSIDE



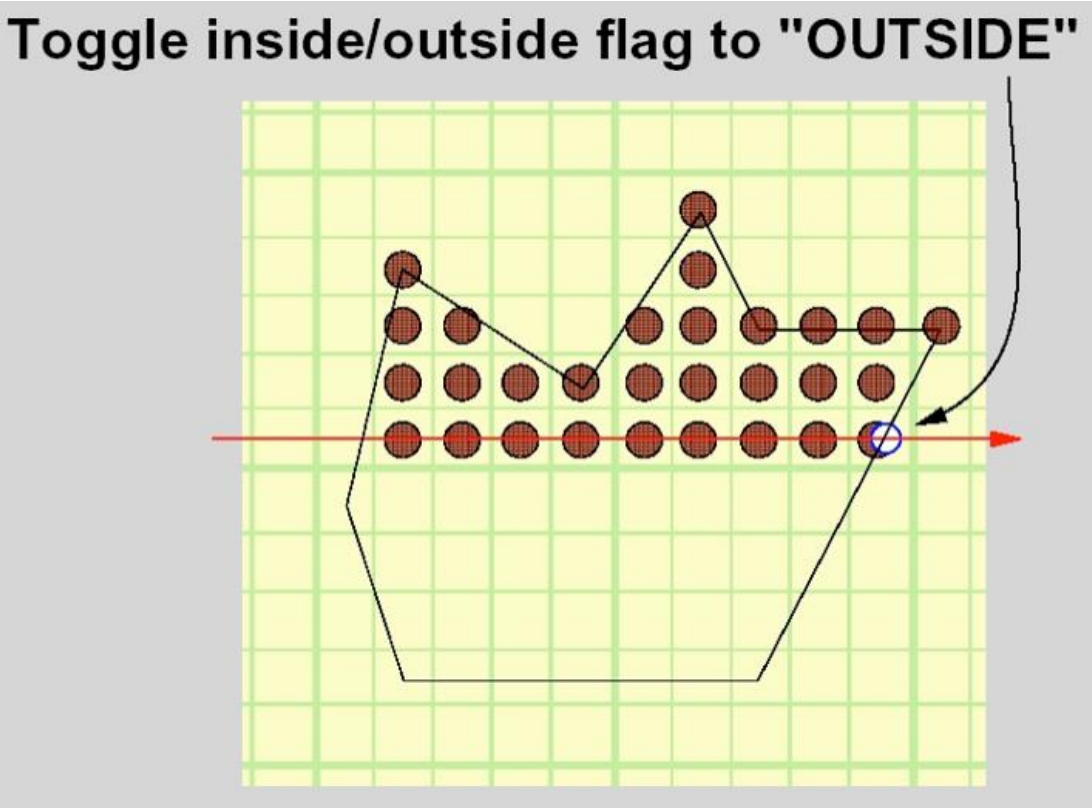
Filled polygon

- We need to determine INSIDE / OUTSIDE



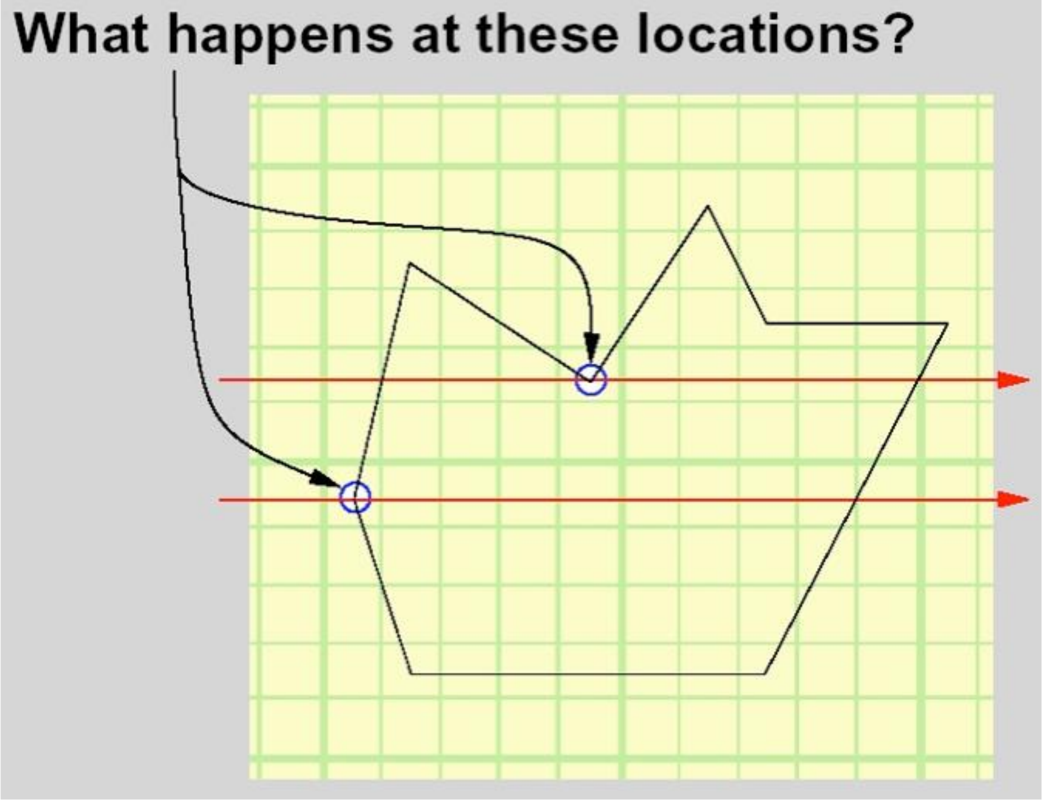
Filled polygon

- We need to determine INSIDE / OUTSIDE



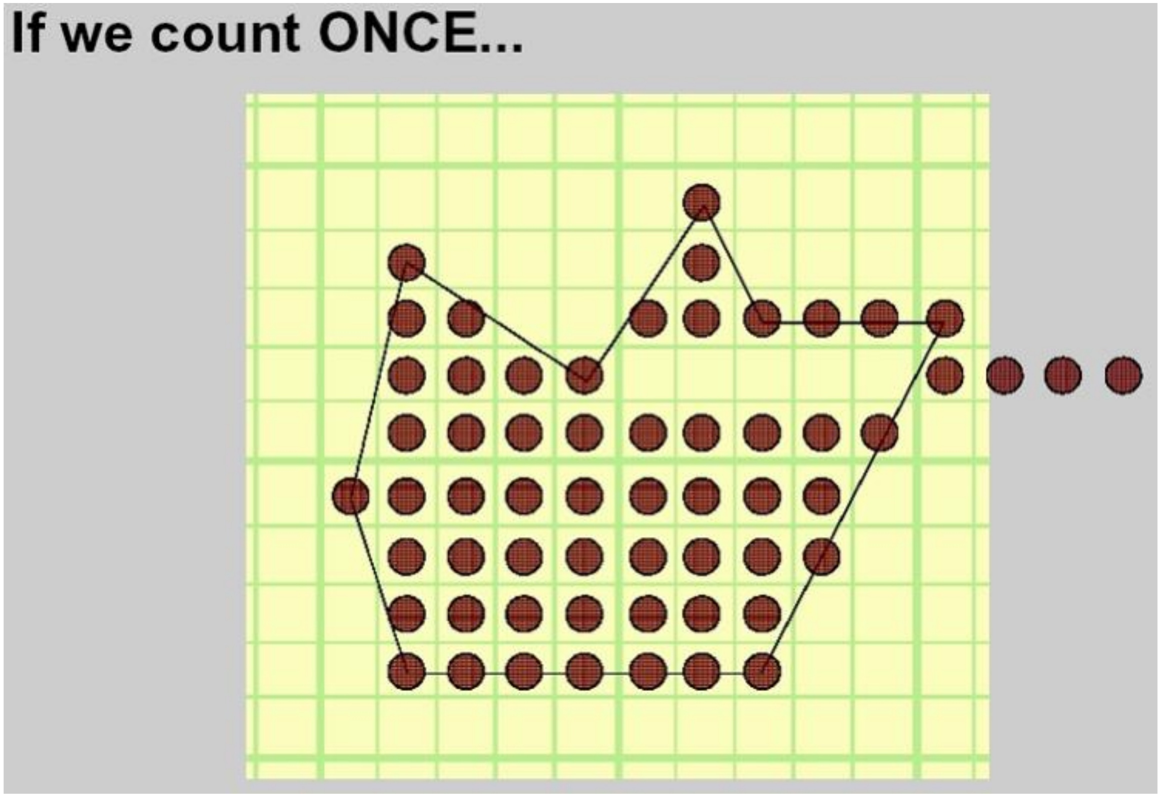
Filled polygon

- We need to determine INSIDE / OUTSIDE



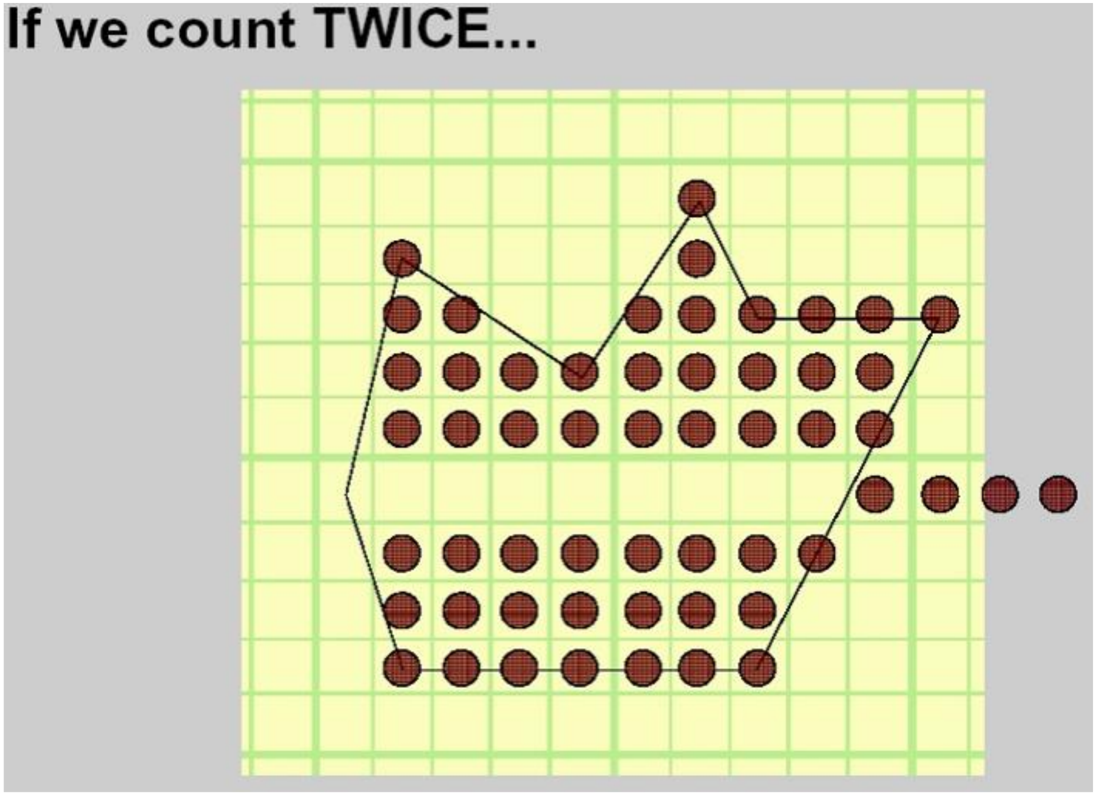
Filled polygon

- We need to determine INSIDE / OUTSIDE



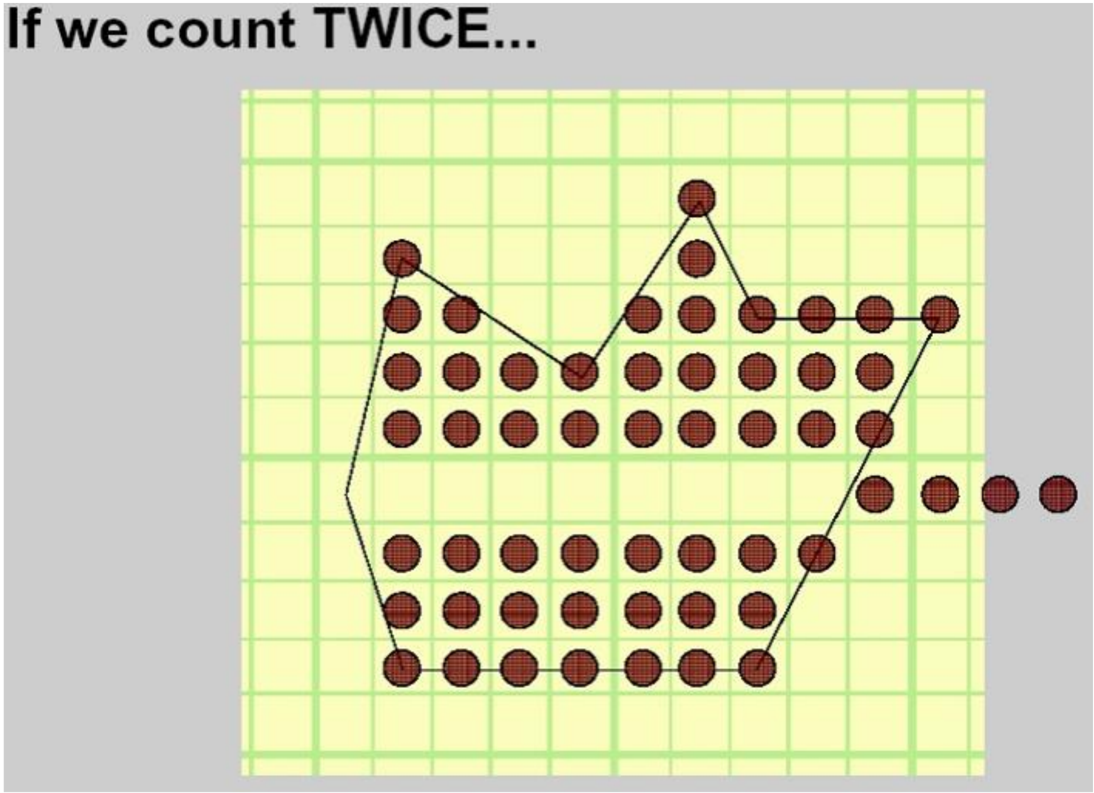
Filled polygon

- We need to determine INSIDE / OUTSIDE



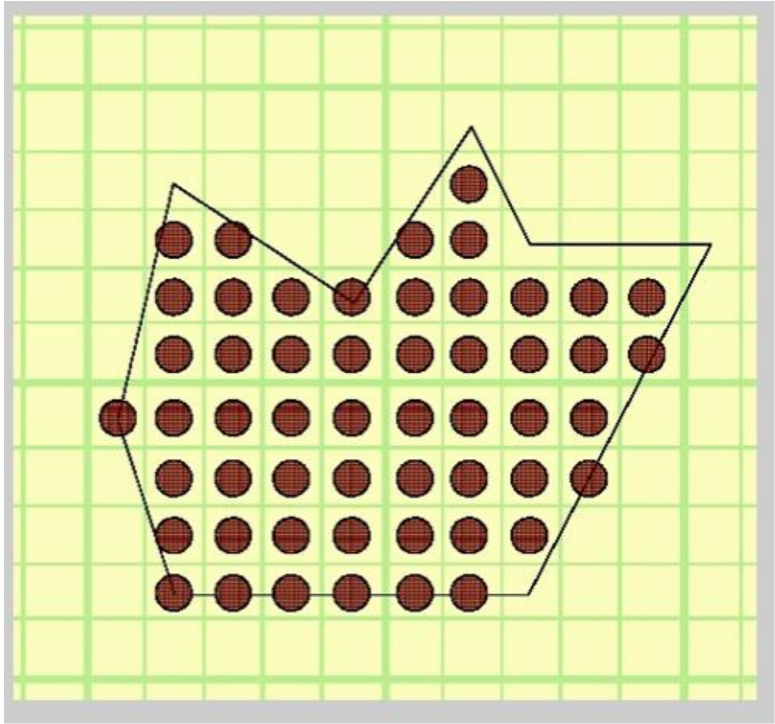
Filled polygon

- We need to determine INSIDE / OUTSIDE



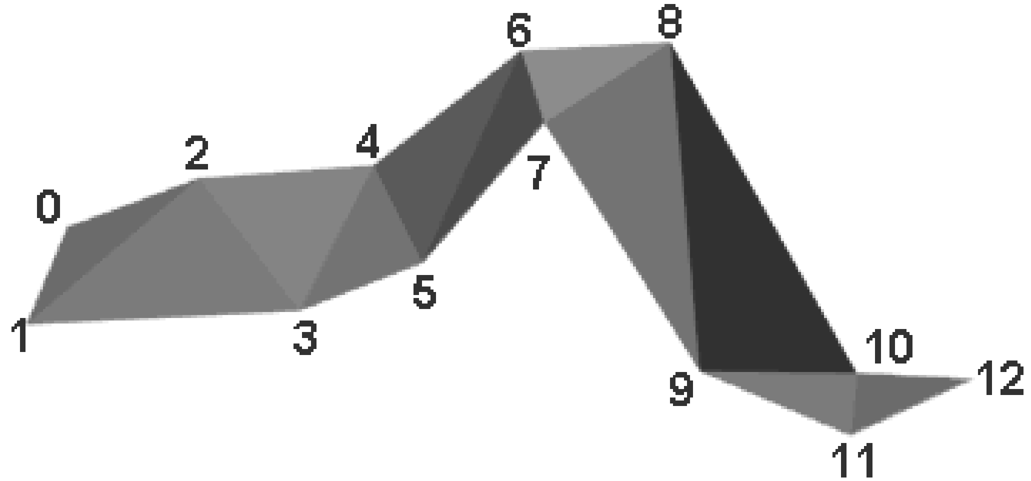
Filled polygon

- If convex / concave vertices are handled correctly



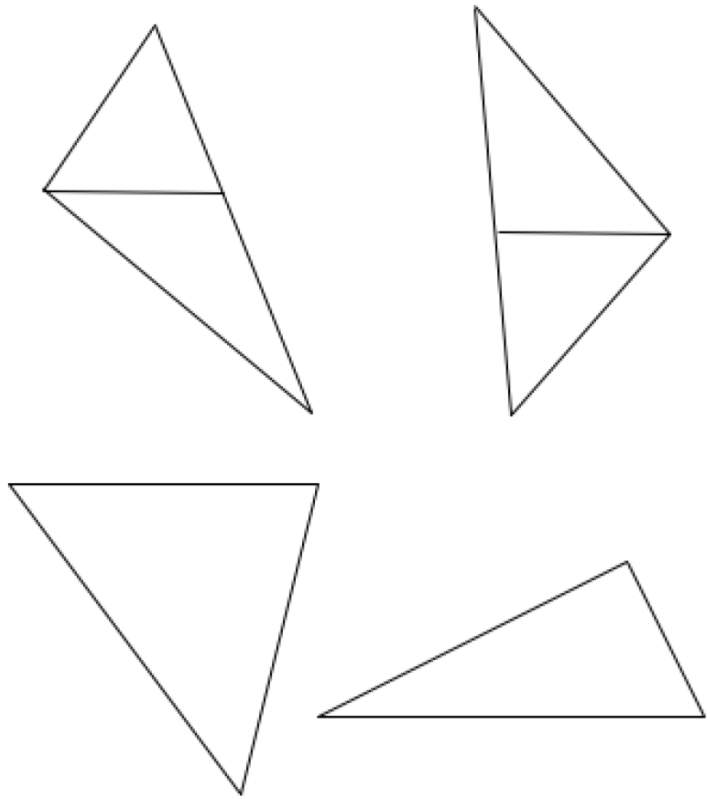
Filled triangle

- Polygons defined using triangles
- Lets draw triangles instead



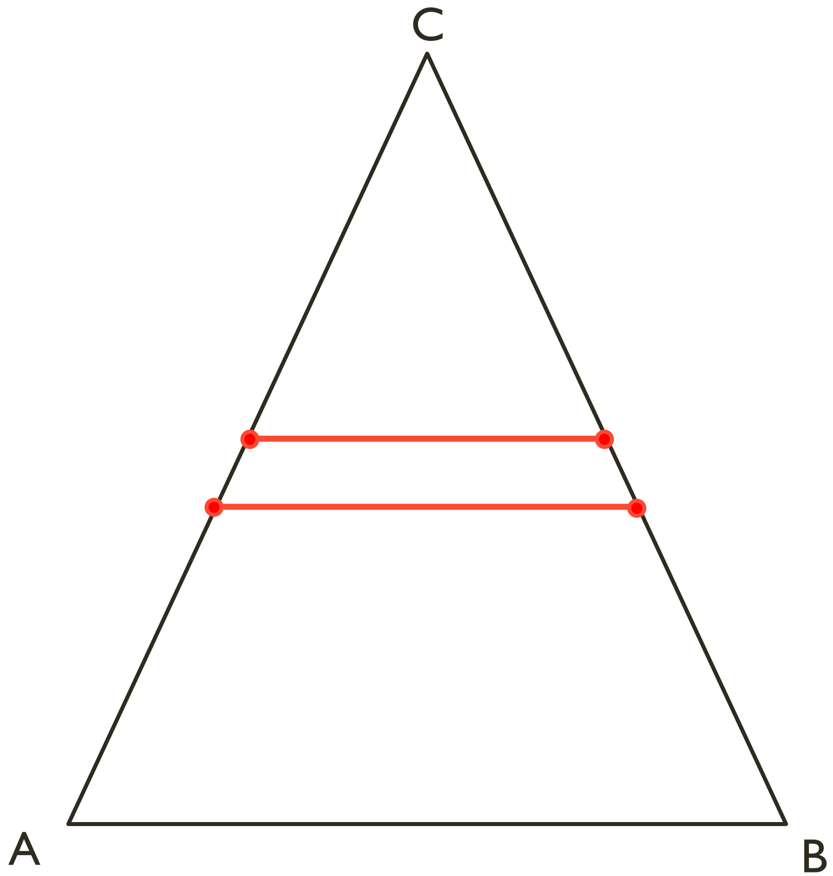
Filled triangle

- Split triangle horizontally into two parts
- Use linear interpolation to draw lines



Filled triangle

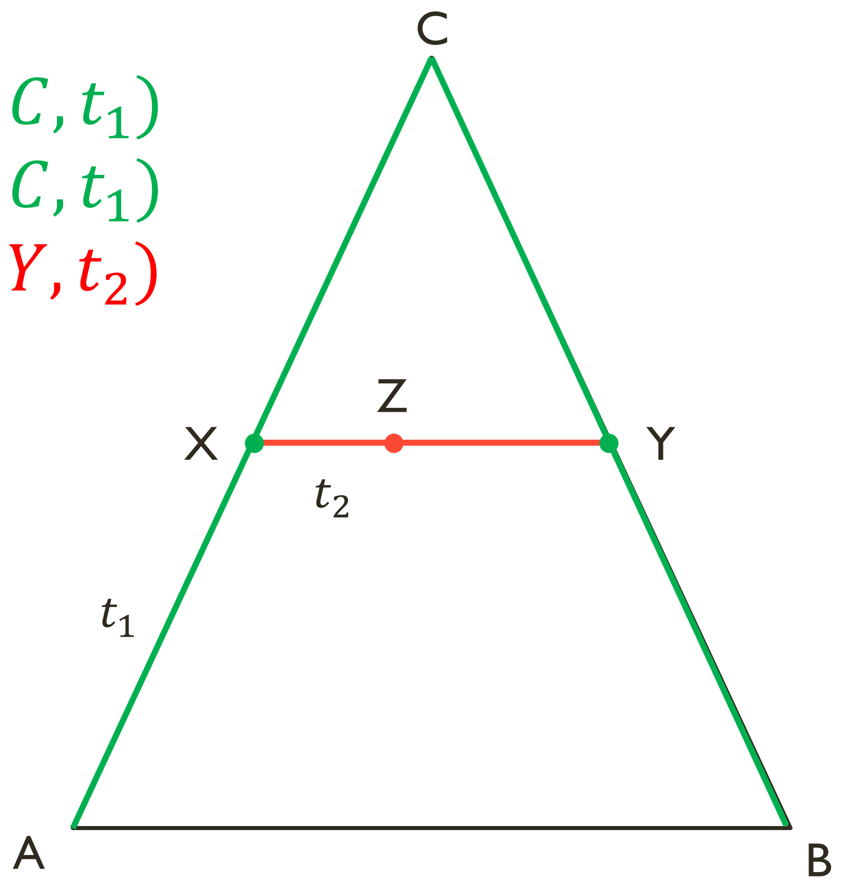
- Fill using horizontal lines



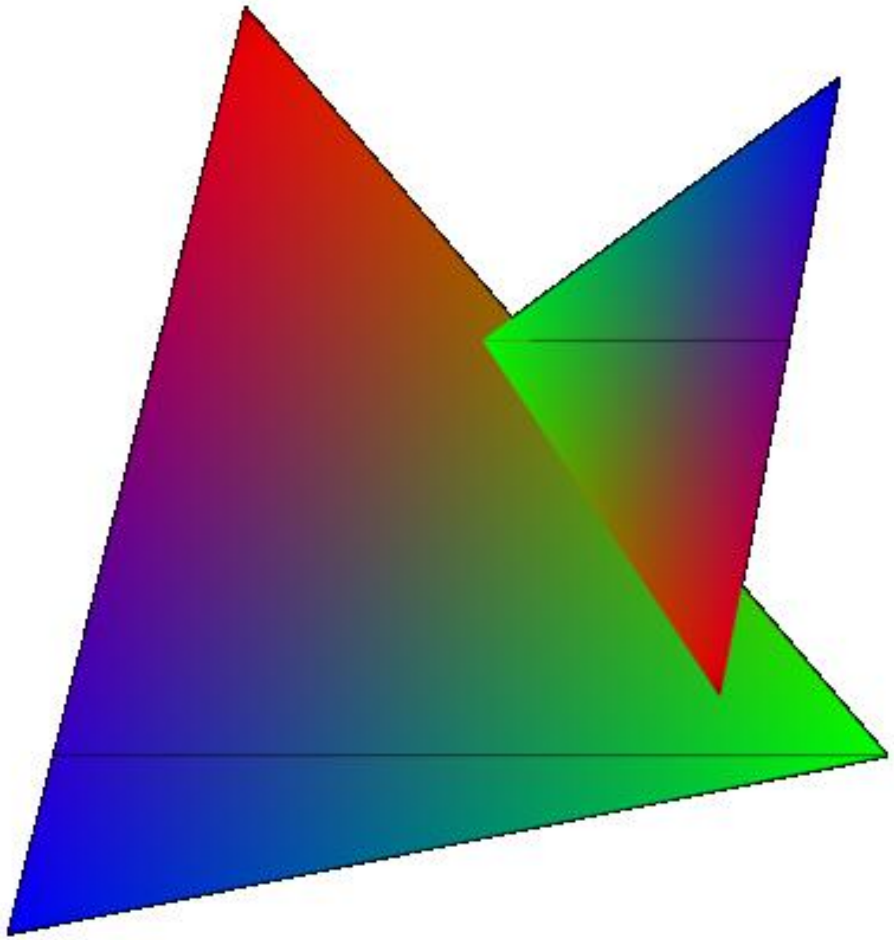
Filled triangle

- Fill using horizontal lines

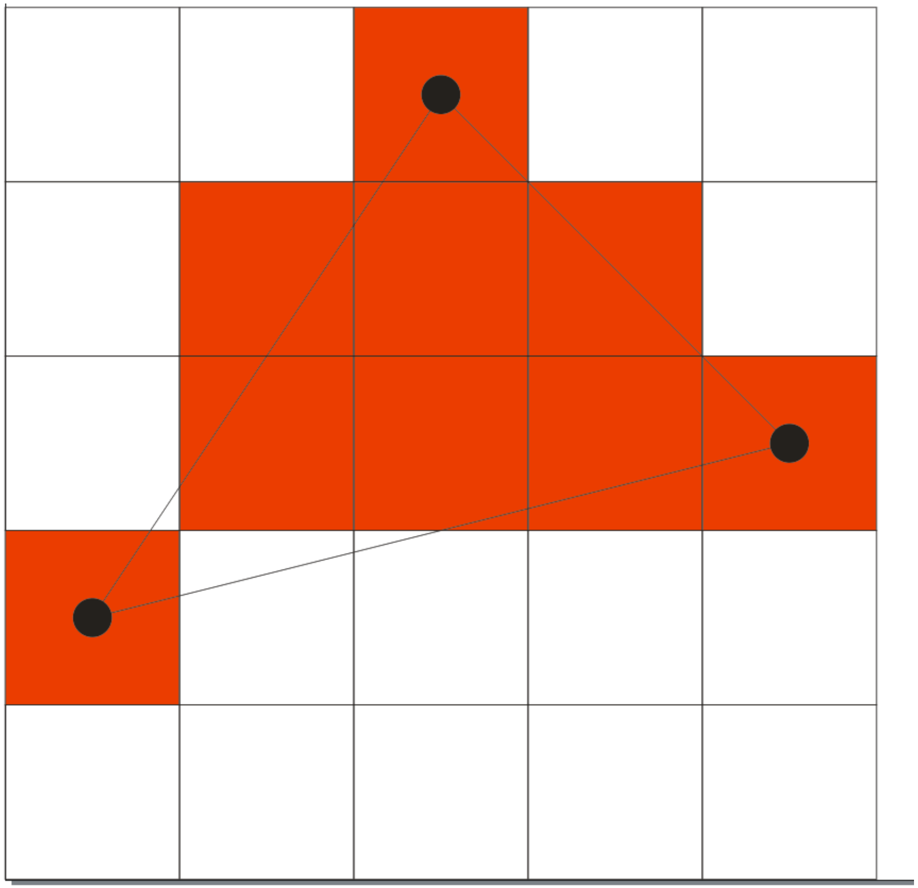
$X = lerp(A, C, t_1)$
$Y = lerp(B, C, t_1)$
$Z = lerp(X, Y, t_2)$



Rasterized triangles

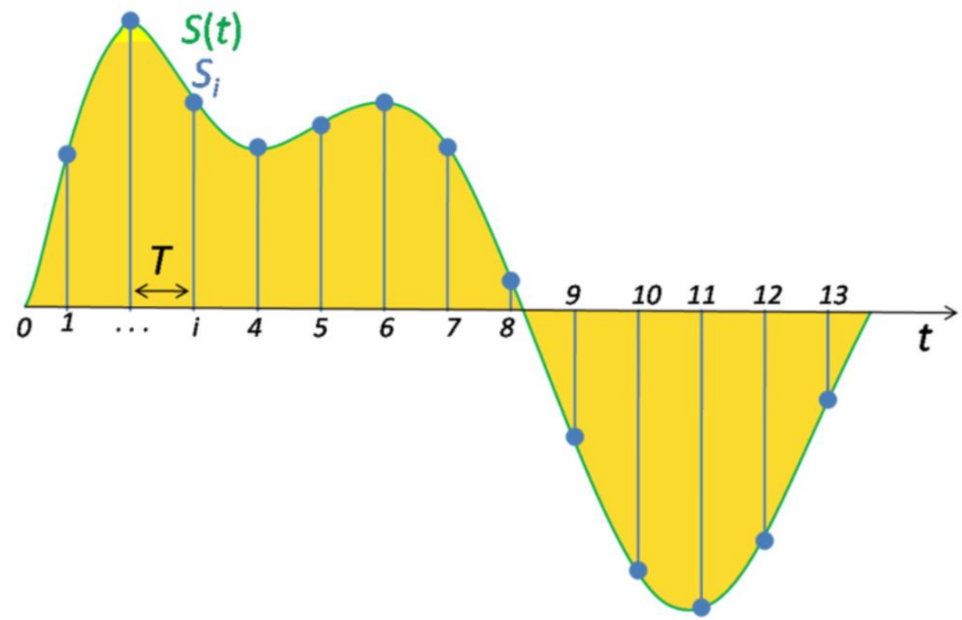


Rasterization alias



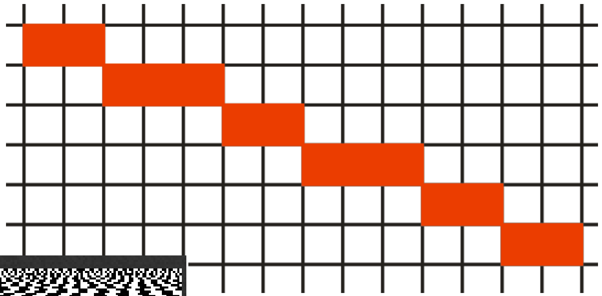
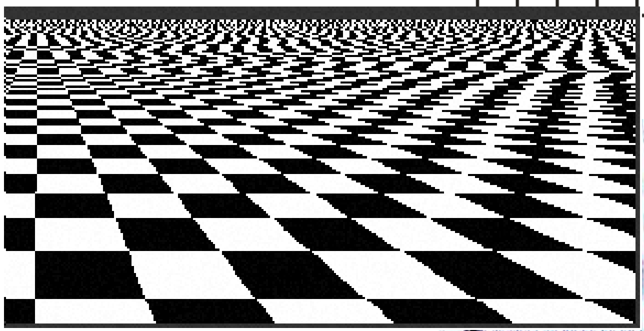
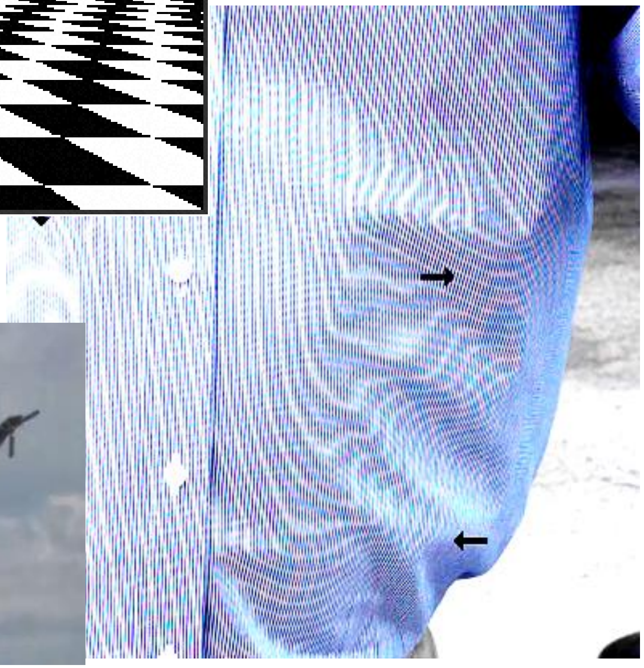
Aliasing

- continuous → discrete: artifacts might appear
- rasterization alias – jagged edges
- sampling
  - creating observation of continuous phenomenon in discrete intervals
- sampling frequency
  - pixel density



Forms of alias

- spatial alias
  - jaggy edges
  - moiré
  - texture distortion
- temporal
  - “wagon wheel”

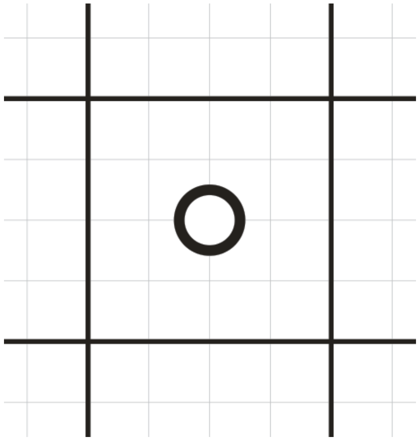

Anti-aliasing

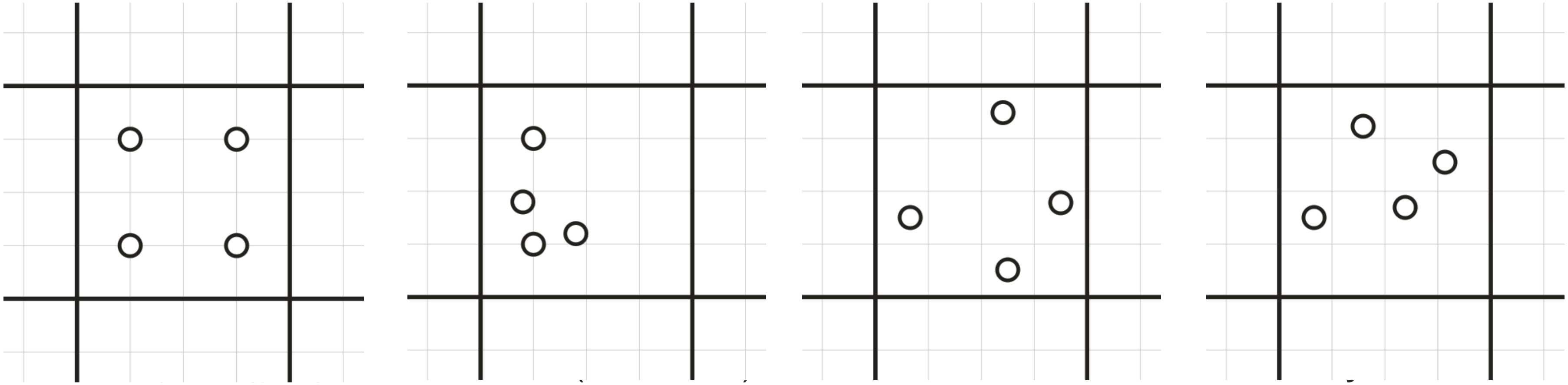
- general (global) anti-aliasing - supersampling
  - works on all objects

- object (local) anti-aliasing
  - line anti-aliasing
  - silhouette anti-aliasing
  - texture anti-aliasing

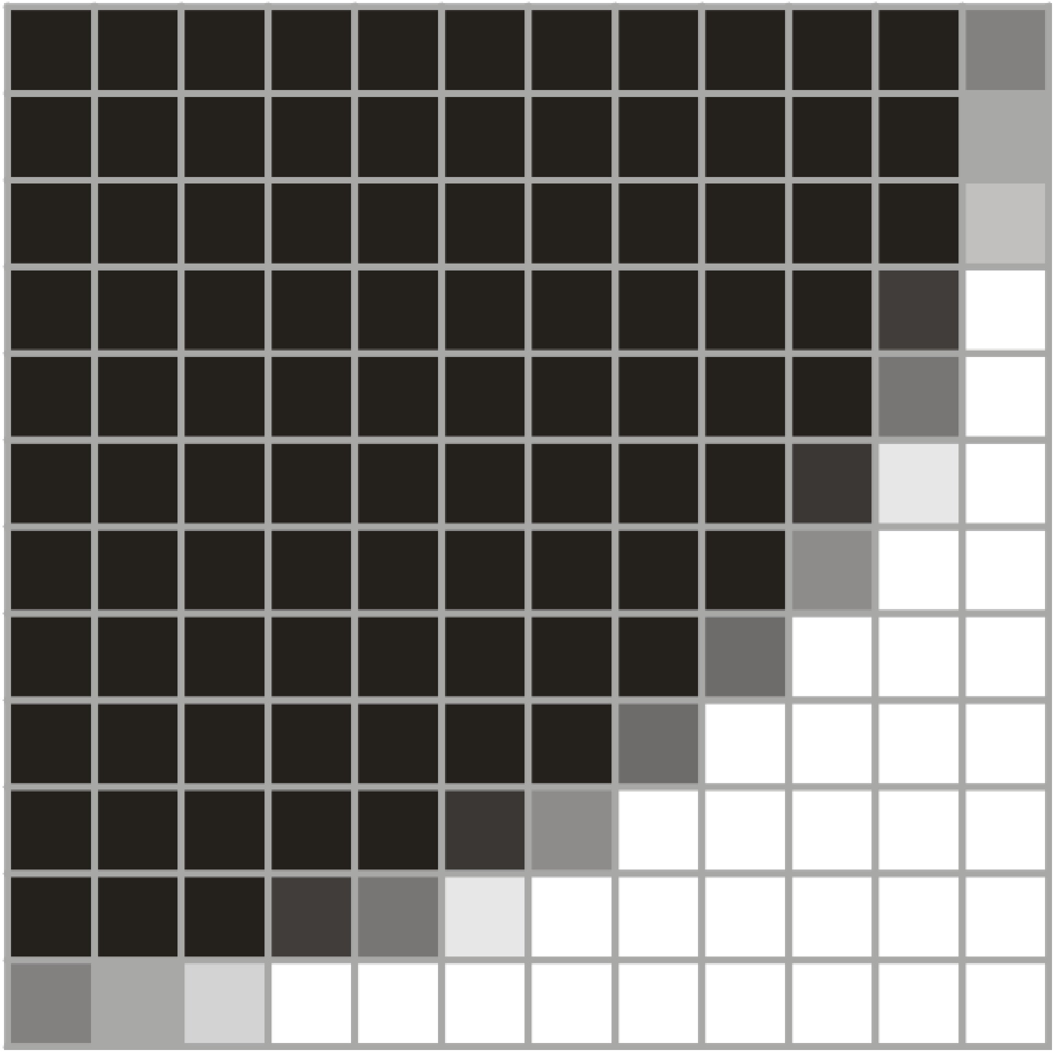
Super-sampling

- For each pixel perform multiple sub-pixel observations and combine the results

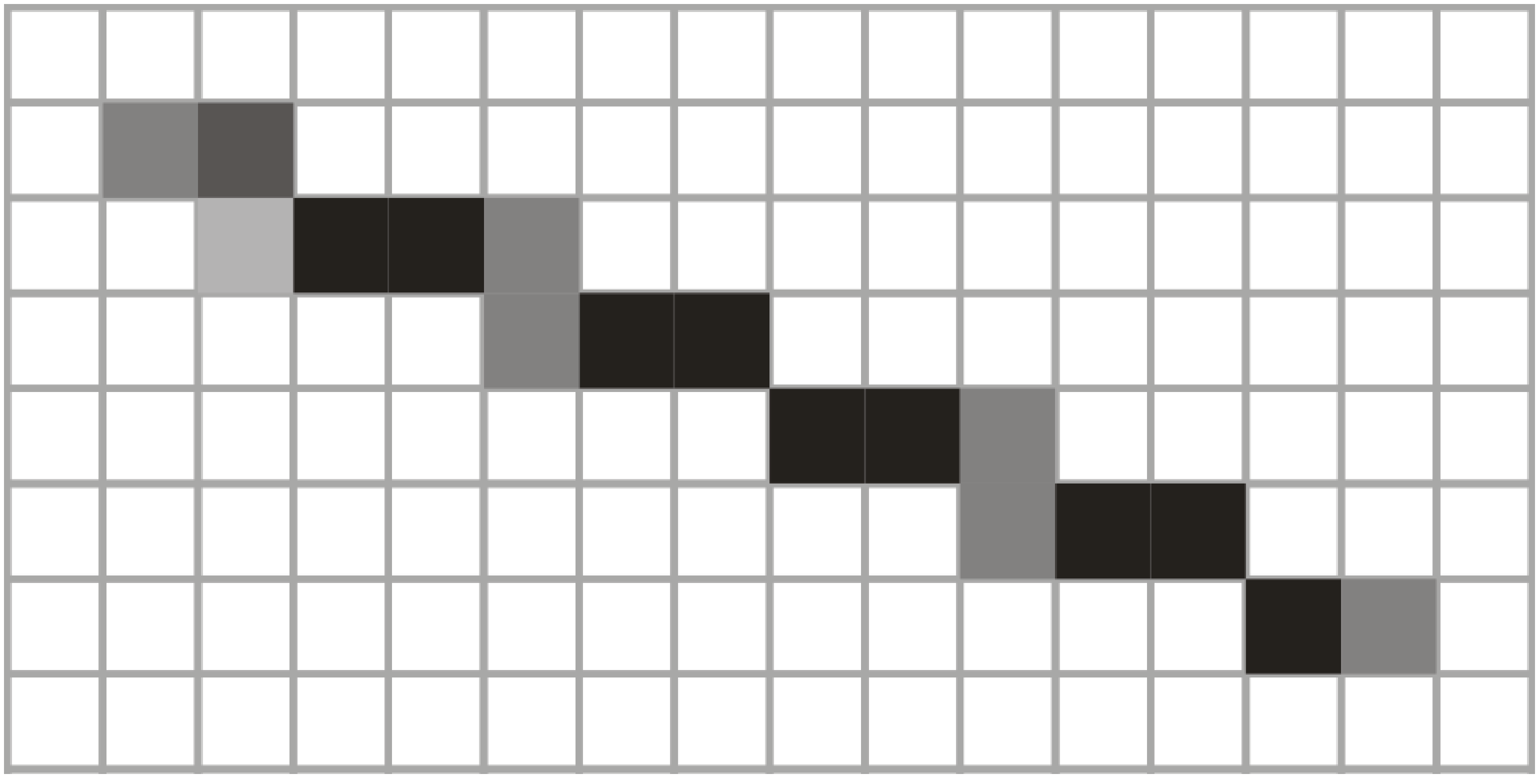




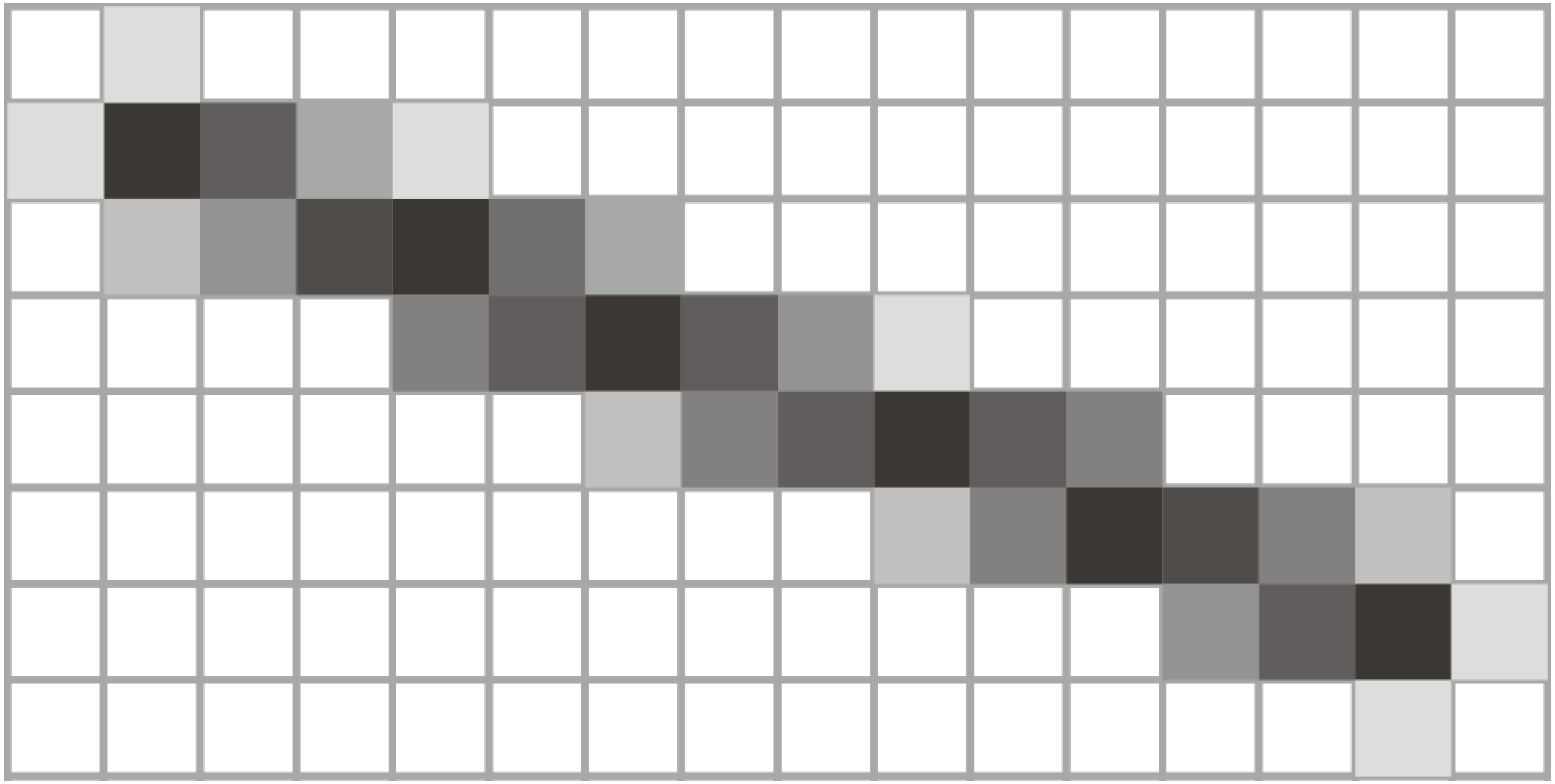
Super-sampling



Super-sampling



Super-sampling



Next lessons

3D polygons

Modeling Transformation

↓

Lighting

↓

Viewing Transformation

↓

Projection Transformation

↓

Clipping

↓

Scan Conversion

2D Image

Shading and Lighting (04)

Rest of rendering pipeline - next lessons

3D polygons

Modeling Transformation

↓

Lighting

↓

Viewing Transformation

↓

Projection Transformation

↓

Clipping

↓

Scan Conversion

2D Image

Visibility, Culling, Cropping (05)

Next Lecture

Shading and Lighting

Acknowledgements

- Thanks to all the people, whose work is shown here and whose slides were used as a material for creation of these slides:

Matej Novotný, GSVM lectures at FMFI UK

Peter Drahoš, PPGSO lectures at FIIT STU

Output of all the publications and great team work

Very best data from 3D cameras

Questions ?!

Skeletex
RESEARCH

www.skeletex.xyz

madaras@skeletex.xyz

martin.madaras@fmph.uniba.sk