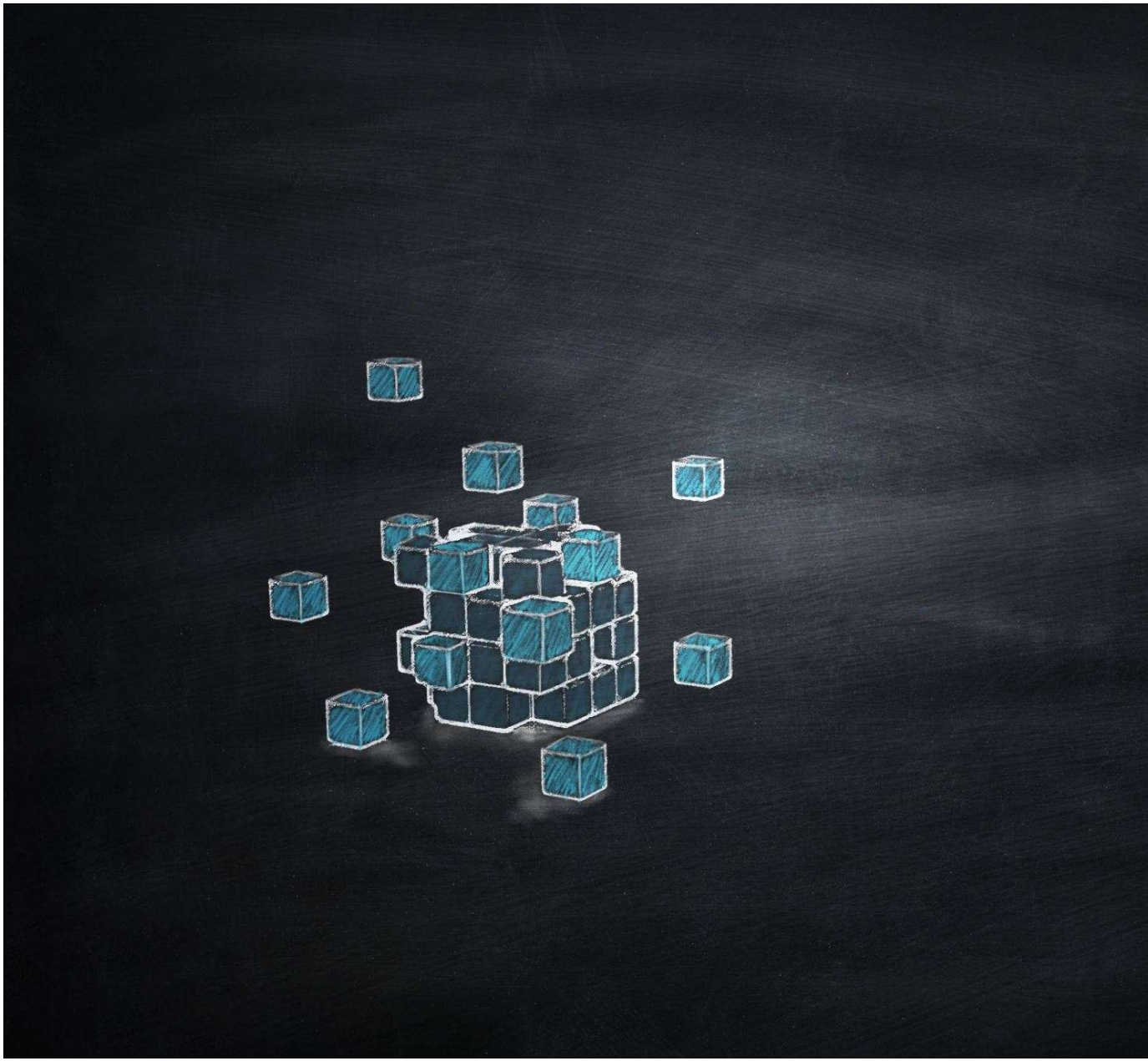
Computer Graphics

Dr. Akram Alsubari

realistic drawing of a cube

- how do we describe the cube?
  - Vertices
  - Edges
- How do we then visualize in 2D?

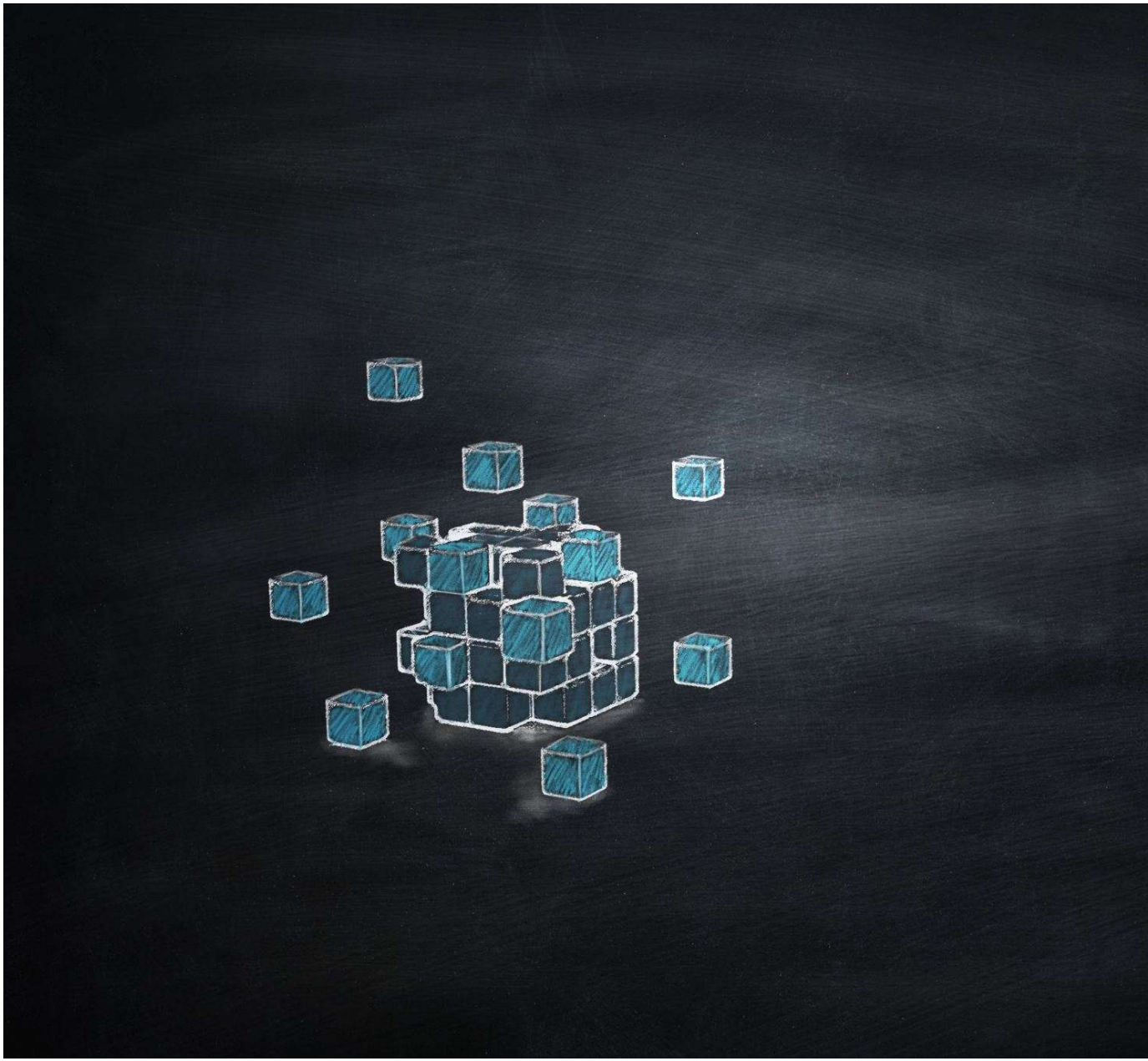


Example for the cube modeling

- Suppose our cube:
  - centered at the origin (0,0,0)
  - dimensions 2 x 2 x 2
- What are the coordinates of the cube vertices and edges?
  - Vertices {A: (1, 1, 1) B: (-1, 1, 1) C: (1,-1, 1) D: (-1,-1, 1) E: (1, 1,-1)F: (-1, 1,-1)G: (1,-1,-1)H: (-1,-1,-1)}
  - Edges {AB, CD, EF, GH,AC, BD, EG, FH,AE, CG, BF, DH}

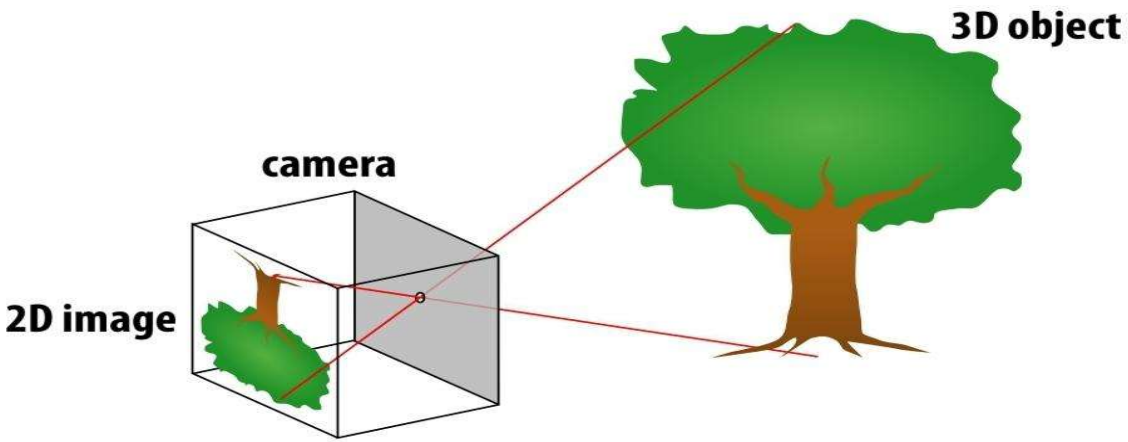
drawing the cube

- How do we draw this 3D cube as a 2D (Flatten) image?

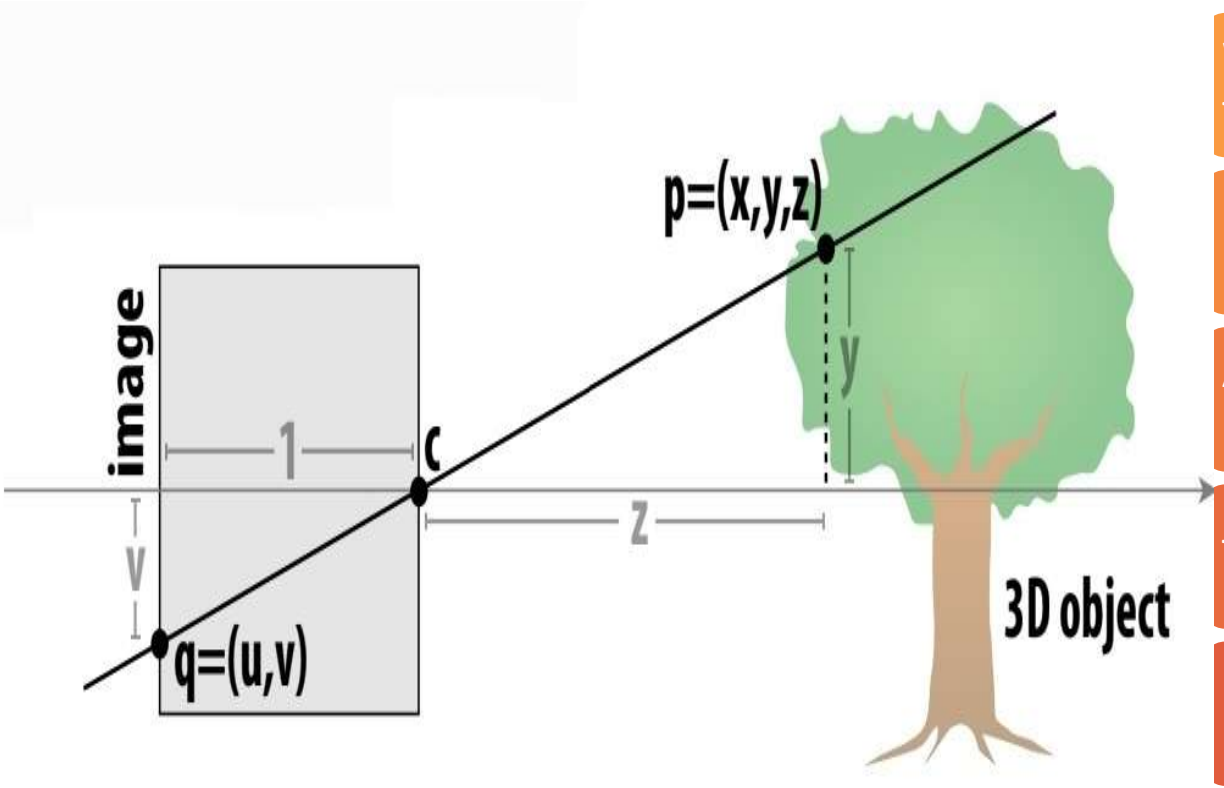


Perspective projection

- Objects look smaller as they get further away (“perspective”)



side view



Where exactly does a point p = (x,y,z) on the tree end up on the image?

Let's call the image point q=(u,v)

Assume camera has unit size, coordinates relative to pinhole c

Then v/1 = y/z… v = y/z

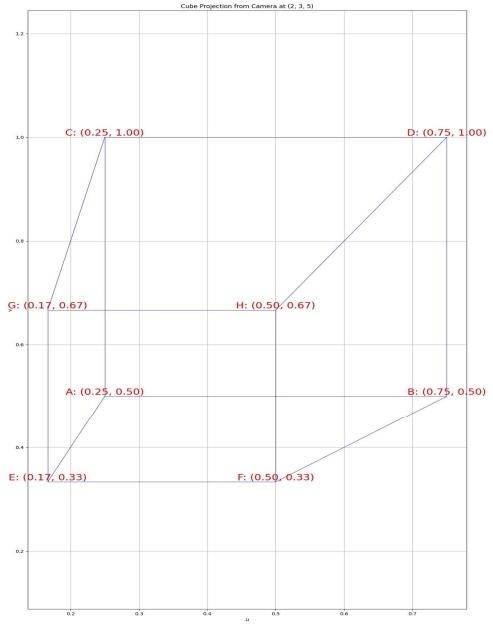
Likewise, horizontal u= x/z

camera position

- Step 1: Define Cube Geometry and Camera
  - Realistic Coordinates 8 points (X,Y,Z)
  - Camera Position (Cx,Cy,Cz)
- Step 2: Compute Screen Coordinates
  - Subtract the camera position: (x,y,z) = (X-Cx, Y-Cy, Z-Cz)
  - Divide x,y by z to get the 2D Coordinates
- Closer to the camera (smaller z), so larger in the projection.
- Farther from the camera (larger z), so smaller

Python Code example

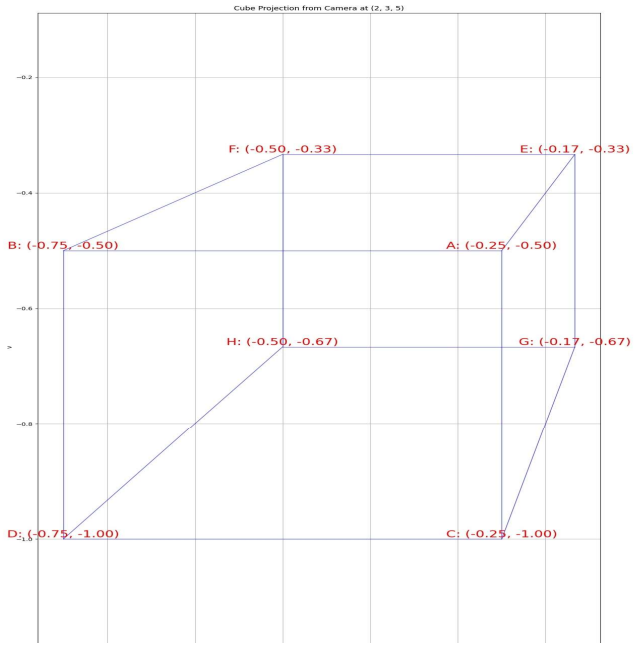
```python
import numpy as np

# Cube definition
vertices = {
    'A': (1, 1, 1),
    'B': (-1, 1, 1),
    'C': (1, -1, 1),
    'D': (-1, -1, 1),
    'E': (1, 1, -1),
    'F': (-1, 1, -1),
    'G': (1, -1, -1),
    'H': (-1, -1, -1)
}

edges = [
    ('A', 'B'), ('B', 'D'), ('D', 'C'), ('C', 'A'),  # Front face (z=1)
    ('E', 'F'), ('F', 'H'), ('H', 'G'), ('G', 'E'),  # Back face (z=-1)
    ('A', 'E'), ('B', 'F'), ('C', 'G'), ('D', 'H')   # Connecting edges
]
def compute_projected_vertices(vertices, camera_pos):
    """Convert world coordinates to screen (u,v) coordinates."""
    projected = {}
    for label, (X, Y, Z) in vertices.items():
        x, y, z = X - camera_pos[0], Y - camera_pos[1], Z - camera_pos[2]
        u = x / z if z != 0 else float('inf')  # Avoid division by zero
        v = y / z if z != 0 else float('inf')
        projected[label] = (round(u,2), round(v,2))
    return projected
camer_pos=(2,3,5)
print(compute_projected_vertices(vertices,camer_pos))
```



{'A': (0.25, 0.5), 'B': (0.75, 0.5), 'C': (0.25, 1.0), 'D': (0.75, 1.0), 'E': (0.17, 0.33), 'F': (0.5, 0.33), 'G': (0.17, 0.67), 'H': (0.5, 0.67)}[Program finished]

Mirror image

- The projection image is mirror, so to resolve the visual multiple Coordinates by -1



Tasks

- Draw the lines between vertical using different lines algorithm

Object segmentation

- Reign Boundaries
- Valleys detection