CS 428: Fall 2009

Introduction to Computer Graphics

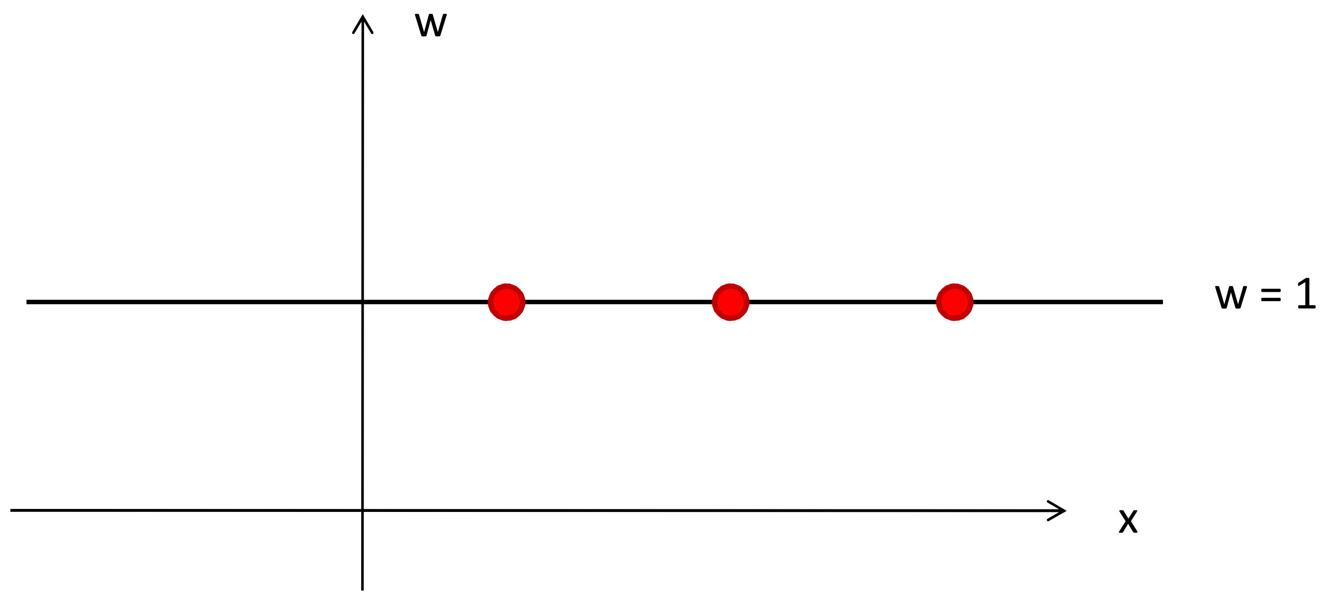
Perspective transformation
some more geometric intuition

Perspective transformation

Geometric intuition

- Shear other axes along w-axis
- For single vanishing point in 2D: shearing the x-axis a w.r.t. w-axis

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{x_0} & 0 & 1 \end{bmatrix}$$

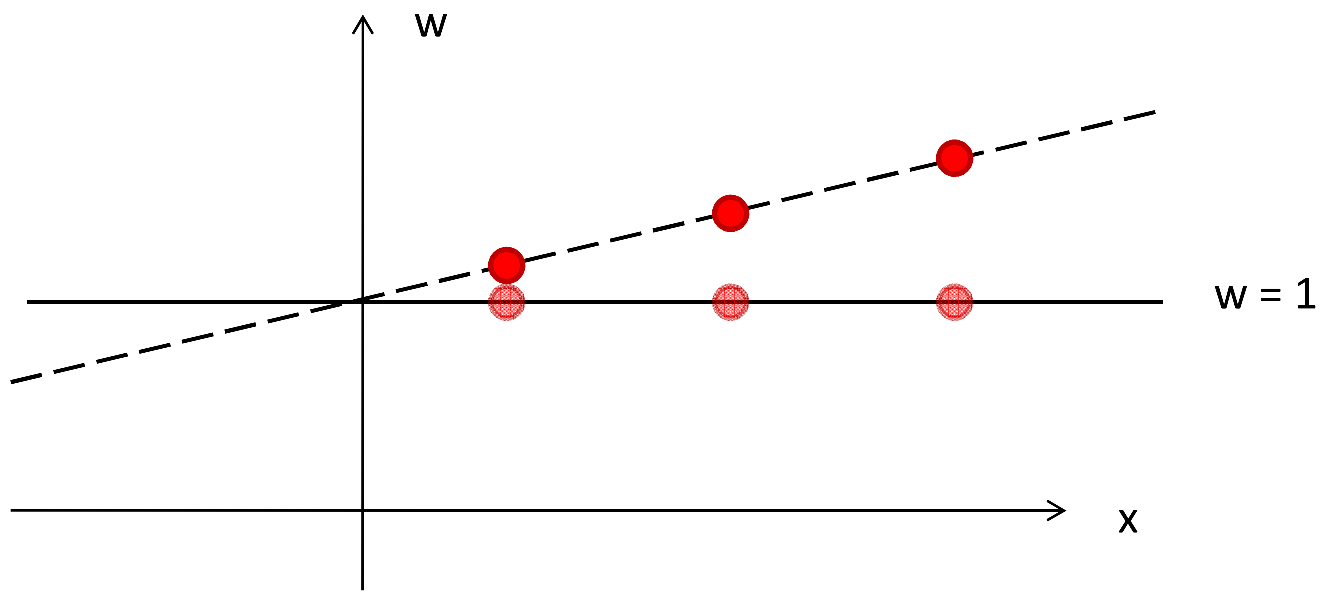


Perspective transformation

Geometric intuition

- Shear other axes along w-axis
- For single vanishing point in 2D: shearing the x-axis a w.r.t. w-axis

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{x_0} & 0 & 1 \end{bmatrix}$$

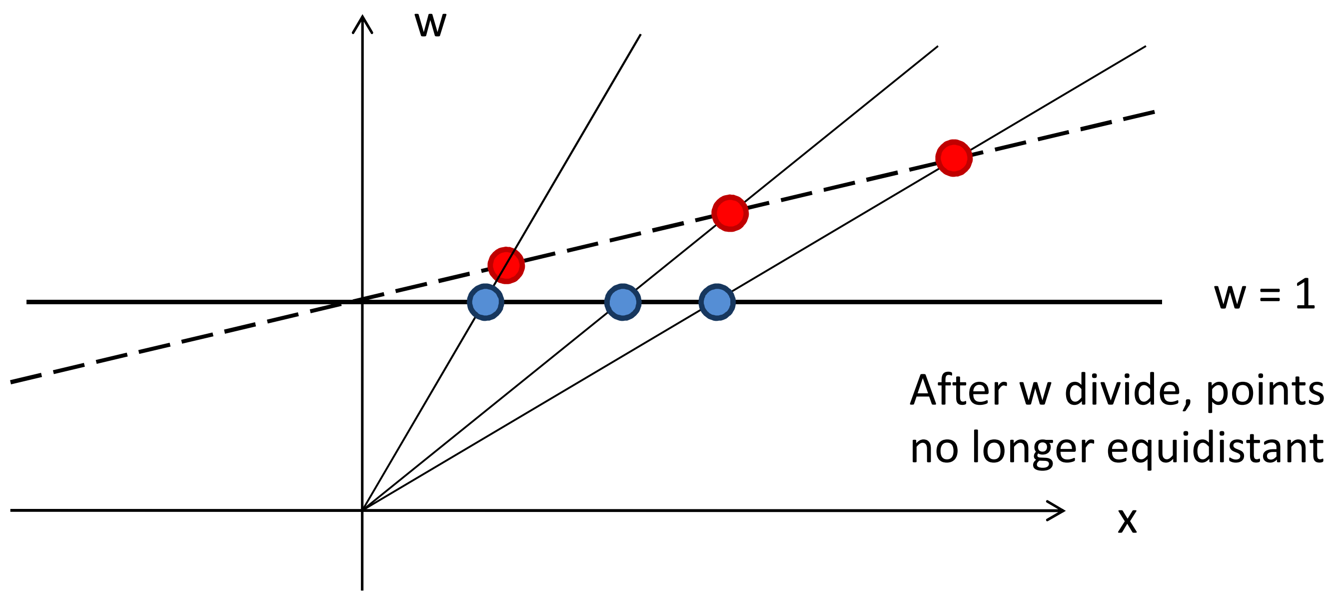


Perspective transformation

Geometric intuition

- Shear other axes along w-axis
- For single vanishing point in 2D: shearing the x-axis a w.r.t. w-axis

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \frac{1}{x_0} & 0 & 1 \end{bmatrix}$$

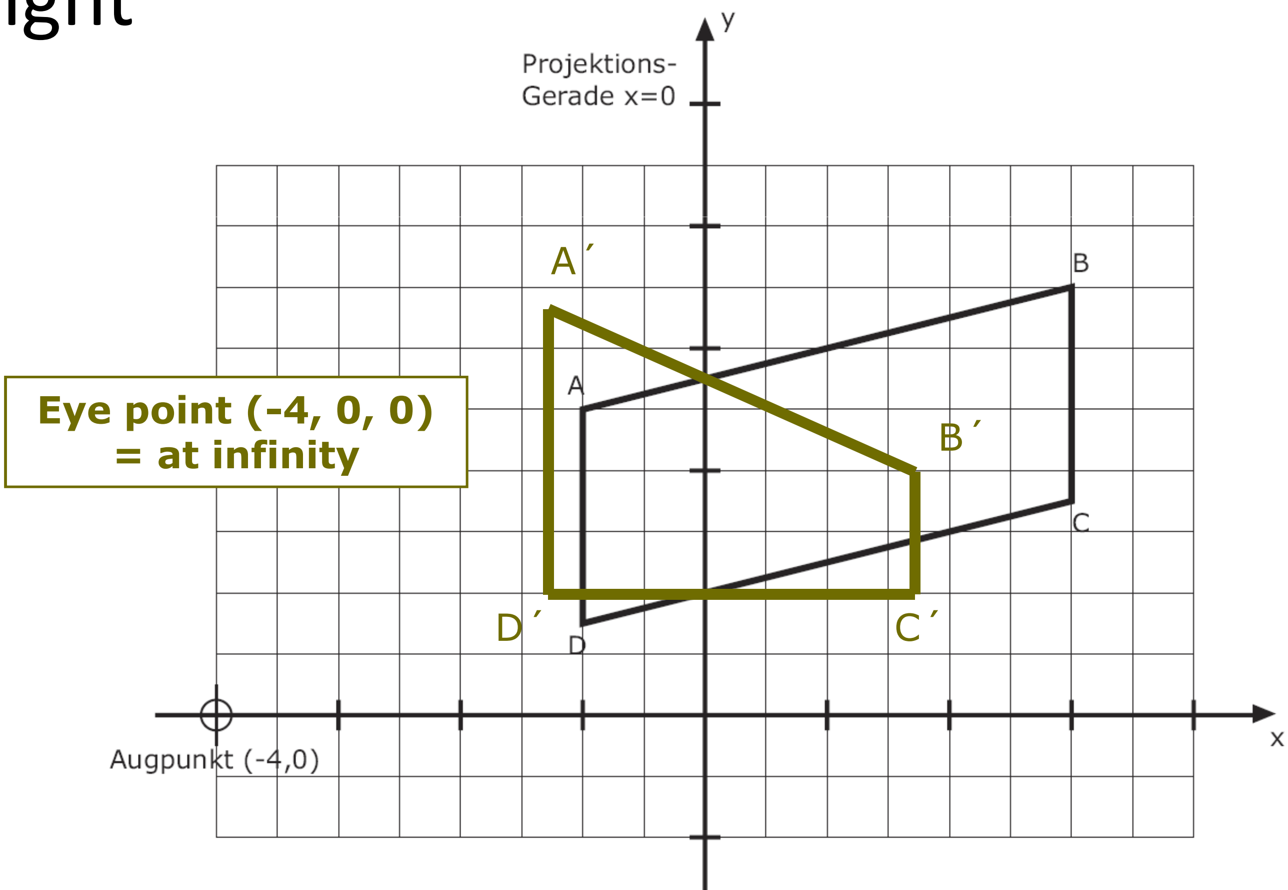


After w divide, points are no longer equidistant

Perspective transformation

Geometric intuition

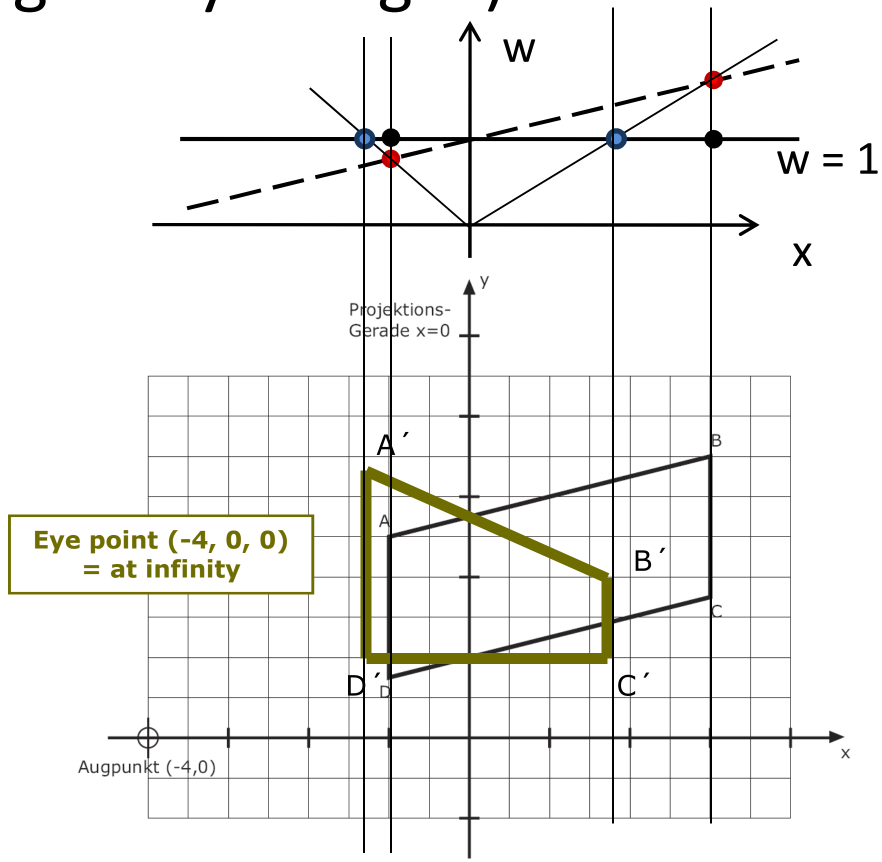
- Geometric construction of **A'B'C'D'** using this insight



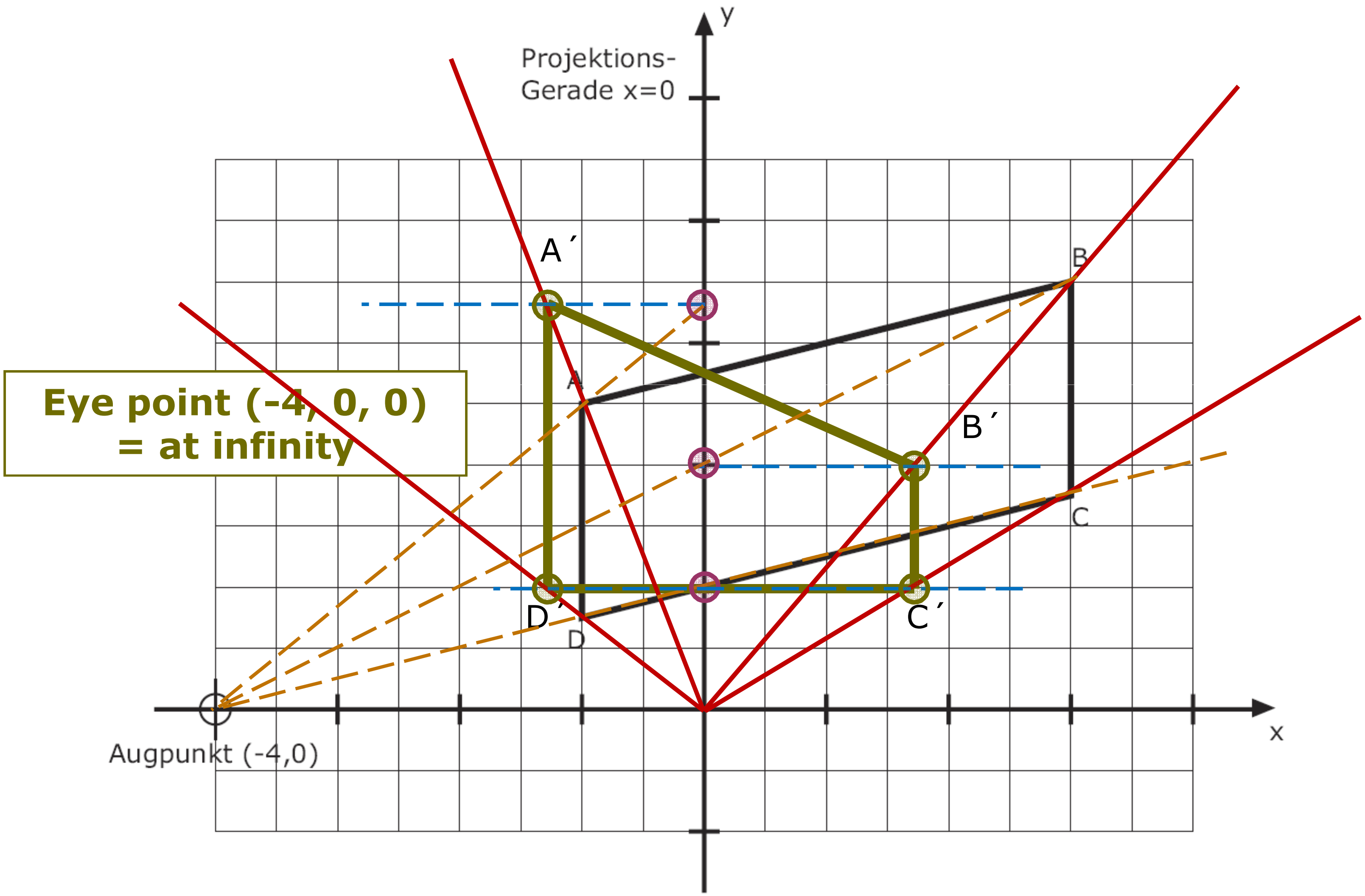
Perspective transformation

Geometric intuition

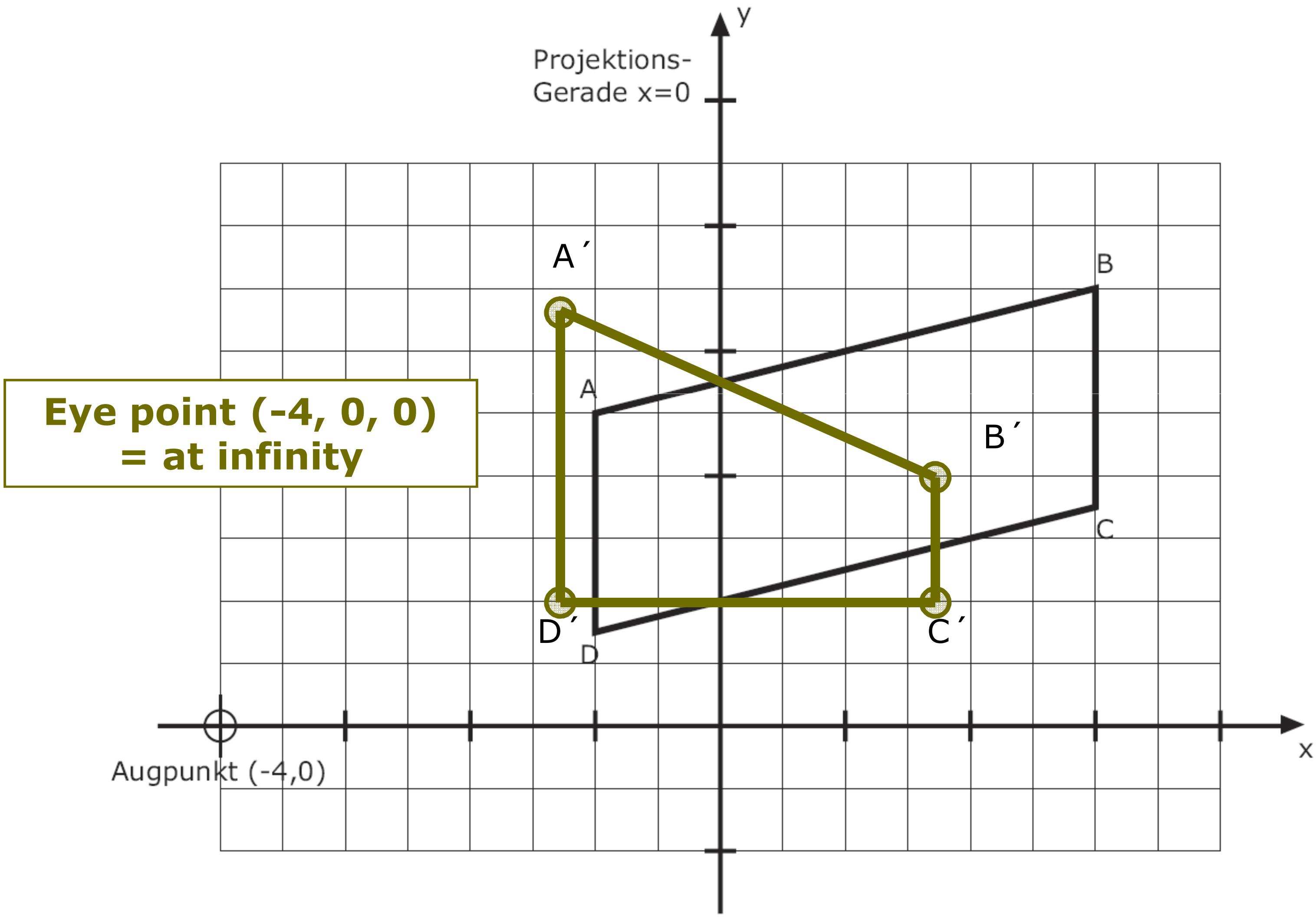
- Shear = translating points **ABCD** in w-direction
  - **ABCD** projects (orthogonally along w) to same polygon after **perspective transformation** (before w-divide!)
  - **ABCD** will no longer lie in w=1 plane
  - w-divide by **central projection A'B'C'D'**



Geometric construction



Geometric construction



CS 428: Fall 2009

Introduction to Computer Graphics

Polygonal meshes

Topic overview

- Image formation and OpenGL
- Transformations and viewing
- **Polygons and polygon meshes**
  - **3D model/mesh representations**
  - **Piecewise linear shape approximations**
  - **Illumination and polygon shading**
- Modeling and animation
- Rendering

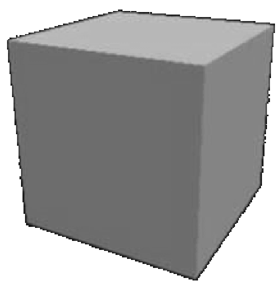
Polygon meshes

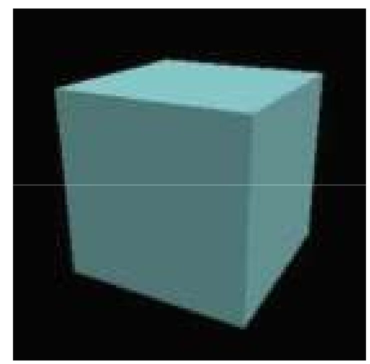
- Some objects are flat
- Some objects are smooth ← **approximate!**
  - Use many **planar** triangles/quadrilaterals to approximate the underlying smooth surface

Approximating shapes with polygons

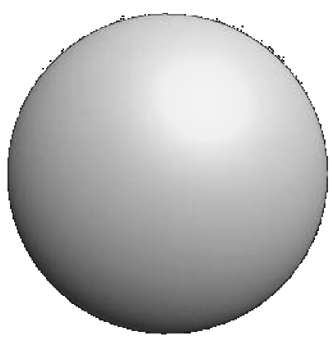
shape

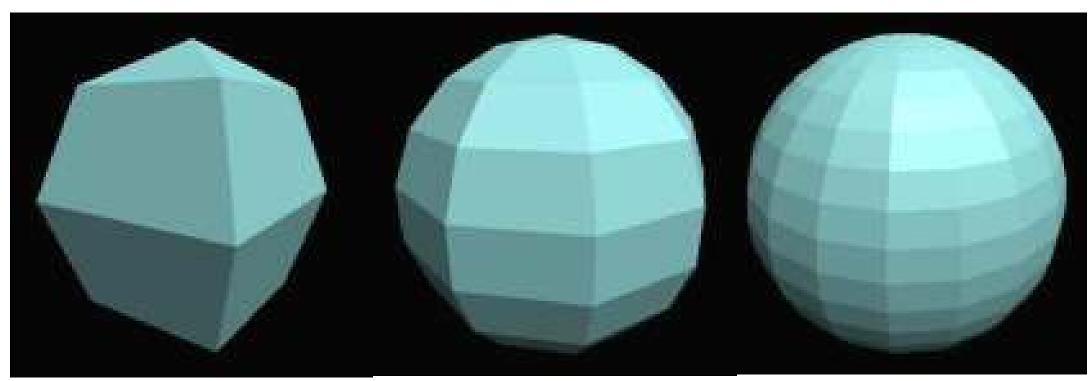
polygon mesh





(exact)

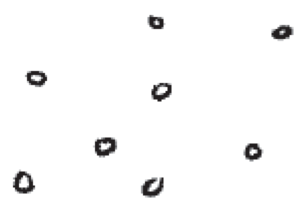




…

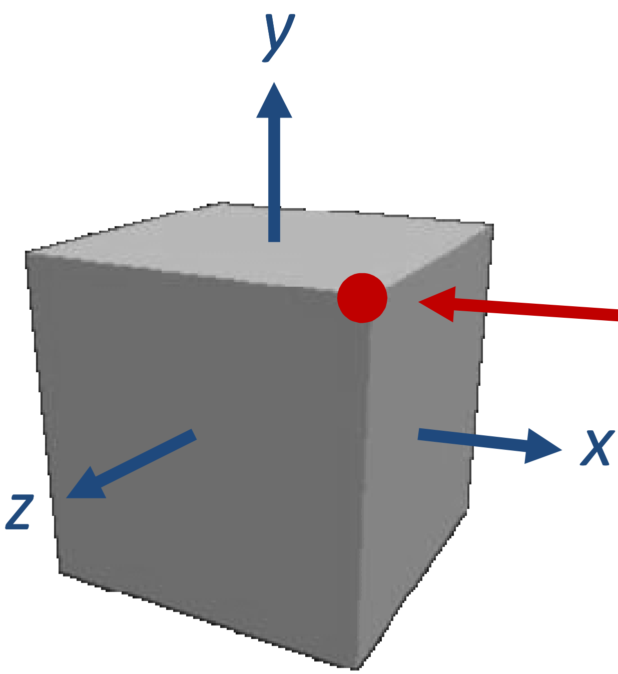
(approximated)

Polygon meshes

- Polygon mesh
  - Vertices  geometry (positions)
  - Edges  topology (connectivity)
  - Faces 
    - All three are redundant, but can lead to more efficient (neighborhood) computation

Representation

- Often just stored in a file
  - List of vertices (x_1, y_1, z_1) … (x_n, y_n, z_n) followed by
  - List of polygons = ordered list of indices (1,2,3) …

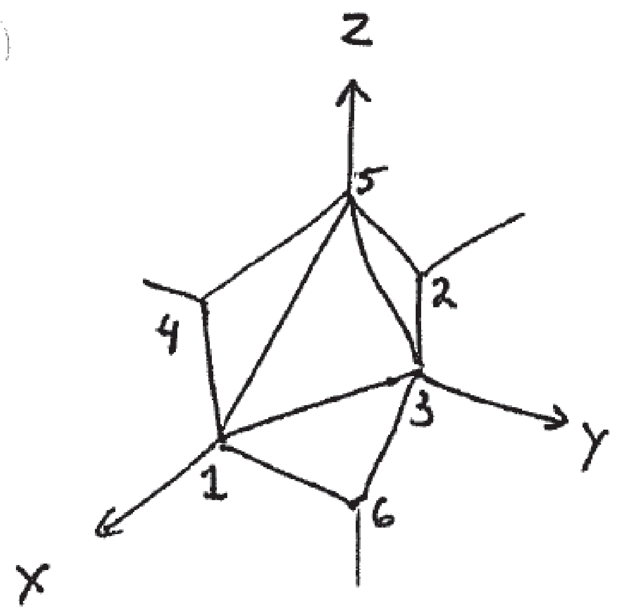


Vertices	Polygons
1 (-1, 1, 1)	{ 1, 2, 3, 4 }
2 (-1, -1, 1)	{ 8, 7, 6, 5 }
3 (1, -1, 1)	{ 4, 3, 7, 8 }
4 (1, 1, 1)	{ 5, 1, 4, 8 }
5 (-1, 1, -1)	{ 5, 6, 2, 1 }
6 (-1, -1, -1)	{ 2, 6, 7, 3 }
7 (1, -1, -1)	
8 (1, 1, -1)	

= Indexed face set

Representation

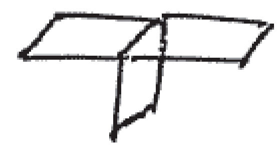
- Example: octahedron



	vertices	polygons
1	(1, 0, 0)	1, 3, 5
2	(-1, 0, 0)	3, 1, 6
3	(0, 1, 0)	4, 1, 5
4	(0, -1, 0)	1, 4, 6
5	(0, 0, 1)	3, 2, 5
6	(0, 0, -1)	2, 3, 6
		2, 4, 5
		4, 2, 6

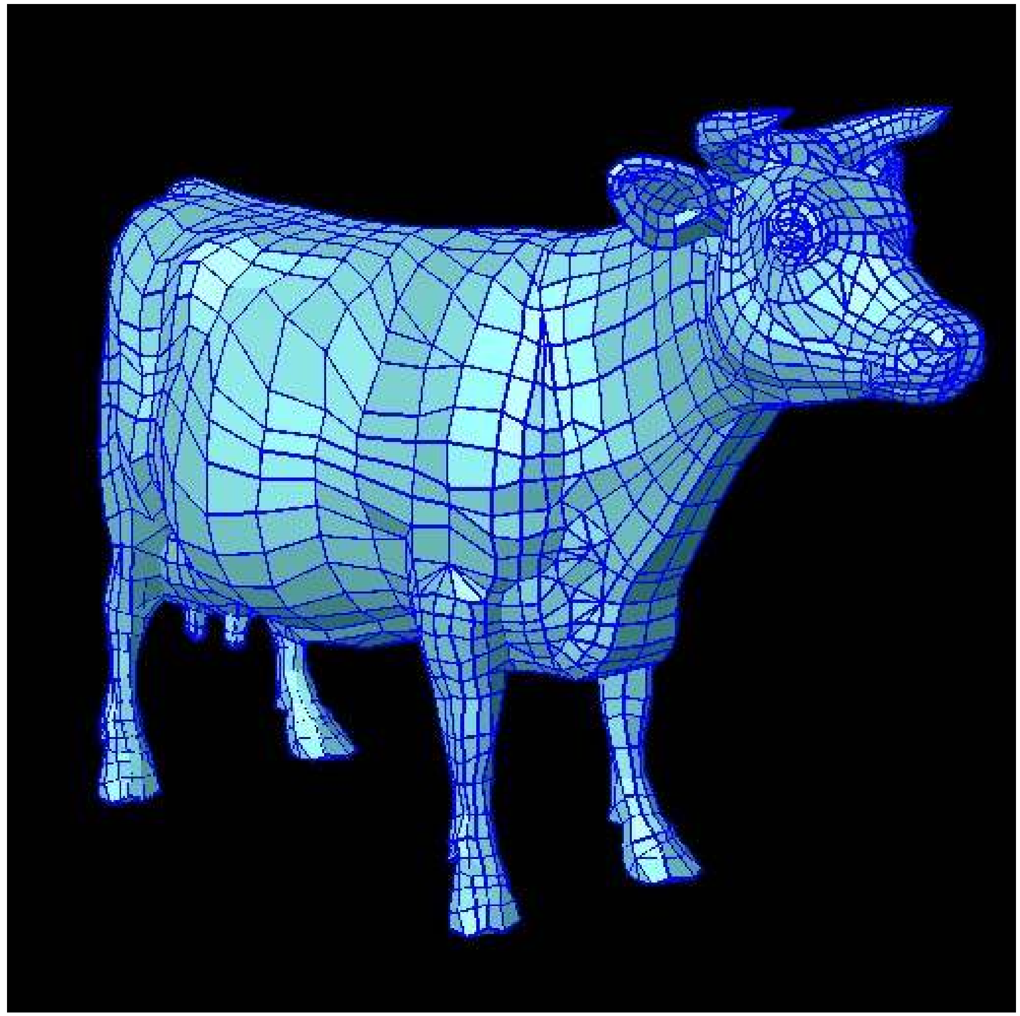
8

Connectivity

- Vertices and polygons are sufficient for rendering
- When adjacency information is needed
  - Edges: 2 vertices
  - 1 or 2 polygons, assuming no T-joins 
  - Vertices store list of adjacent vertices, edges or polygons
  - Polygons store list of edges
- Sophisticated data structures exist (CS 523)

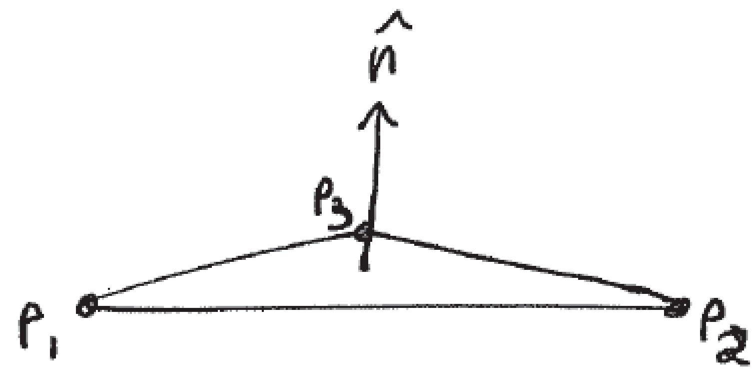
Polygon mesh example

2903 vertices
3263 polygons



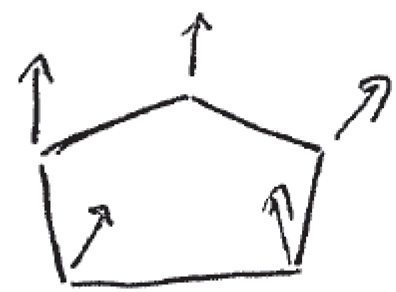
Polygon normals

- Triangles have a single normal vector



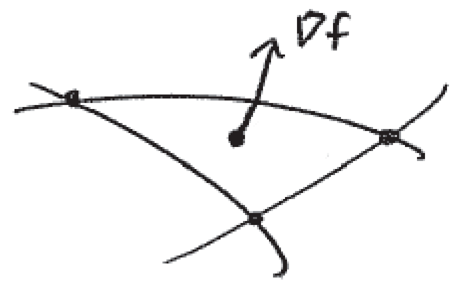
$$n = (p_1 - p_2) \times (p_1 - p_3)$$

- More than 3 points produces a normal at each vertex
  - If all points in a plane all normals are equal

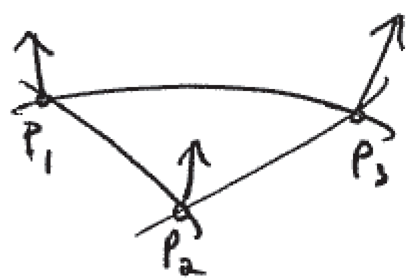


Polygon normals

- If the polygon is sampled from a surface, we can compute normals analytically
  - Disance field f(x,y,z) = 0 … a map from $R^3 \rightarrow R$
  - The gradient ∇f is the (un-normalized) normal at (x,y,z)

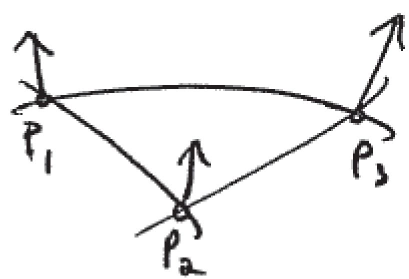


  - But we can find the normals at the vertices here
    - How?



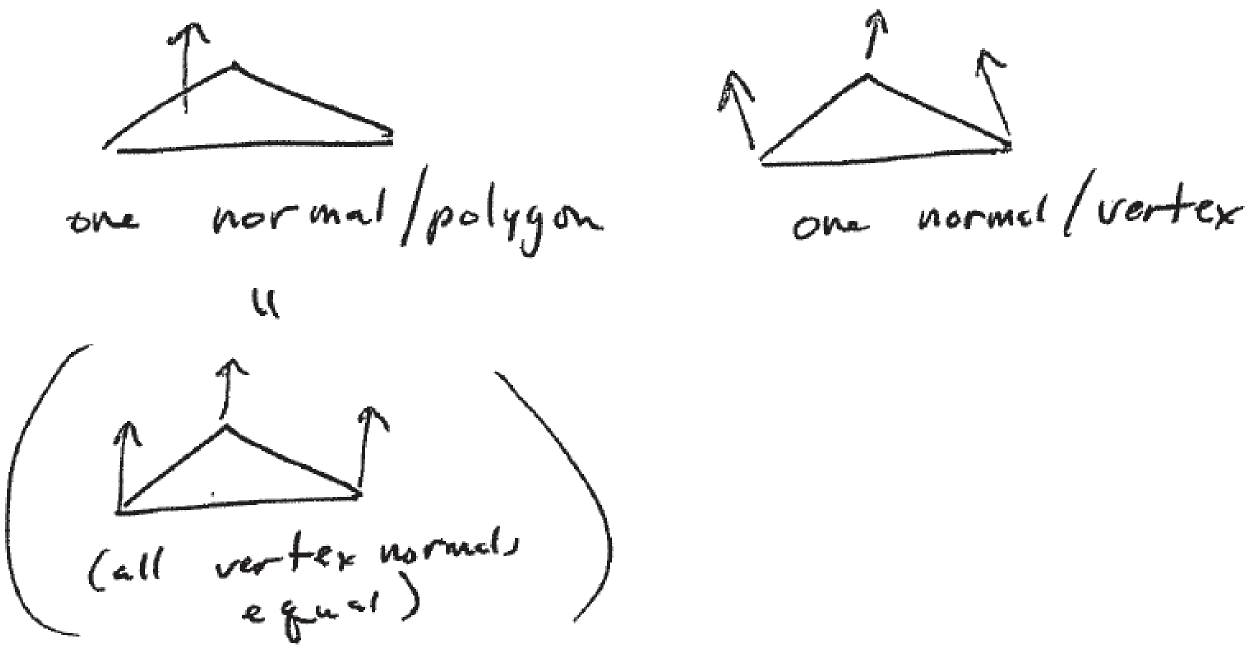
Vertex normals

- Average the normals of adjacent polygons
- For an arbitrary vertex
  - Compute the cross product between each two adjacent outgoing edges (= each adj. polygon)
  - Sum the resulting vectors into a single vector
  - Normalize this vector
- More sophisticated methods exist (CS 523)



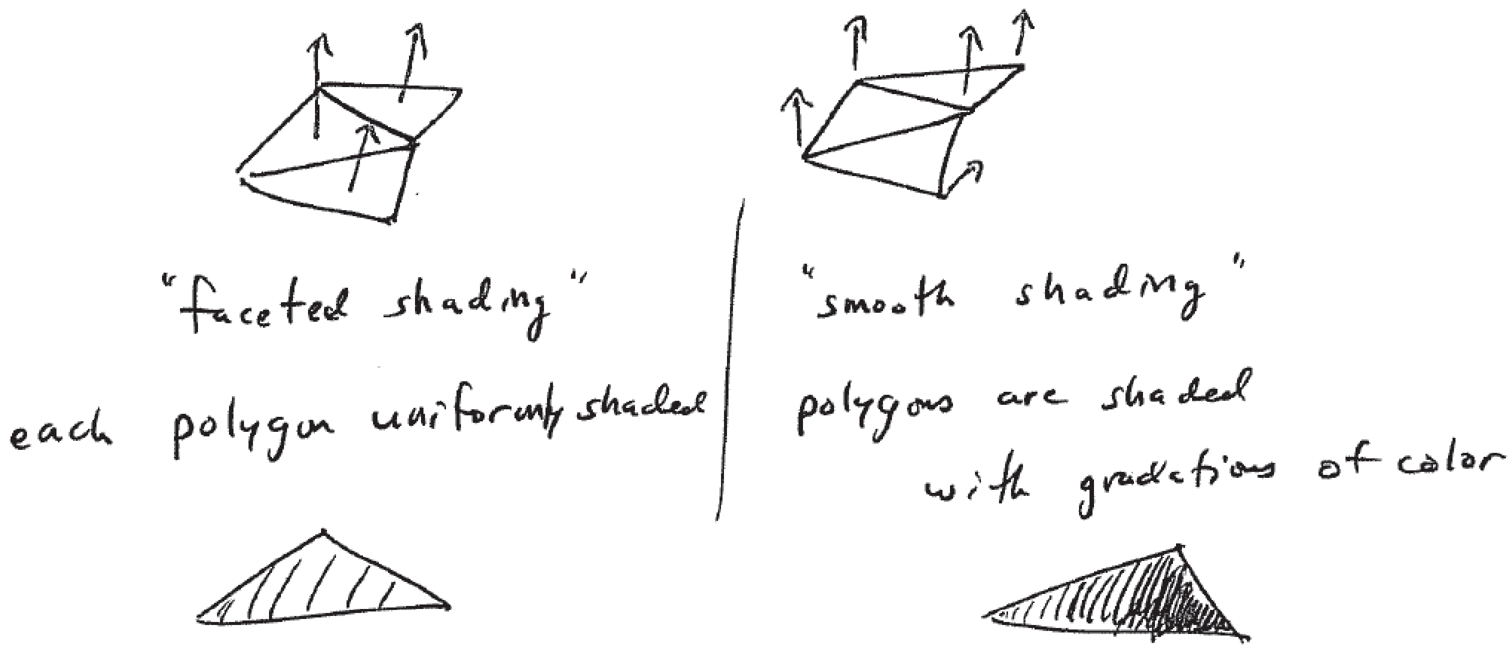
Polygon shading

- For now (more details later): normals are used for **shading** (= computing brightness values)
- One polygon



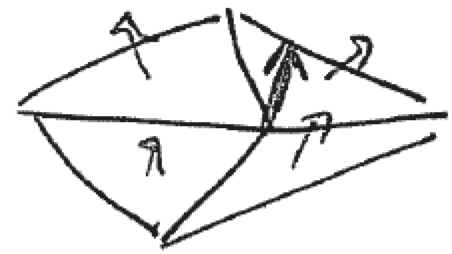
Polygon shading

- For now (more details later): normals are used for **shading** (= computing brightness values)
- Multiple polygons



Smooth shading

- Find average normal of adjacent polygons



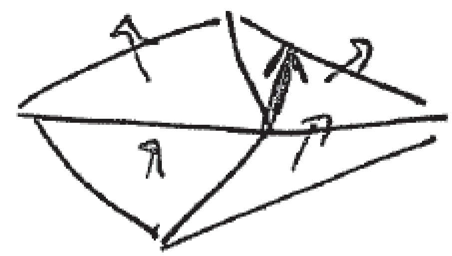
- How to compute?

$$\hat{n}_{avg} = \left(\sum_{\substack{i \in \text{adj}' \\ \text{polygon}}} \hat{n}_i \right)$$

normalize

Smooth shading

- Find average normal of adjacent polygons
- Do we need a list of adjacent polygons?
  - Not if we want to compute **all** avg. normals
- This can be performed from an indexed face set on reading the file



[compute $\hat{n}_i$ for each face

$\forall$ nodes j

$\quad (\hat{n}_{avg})_j = 0$

$\forall$ faces k

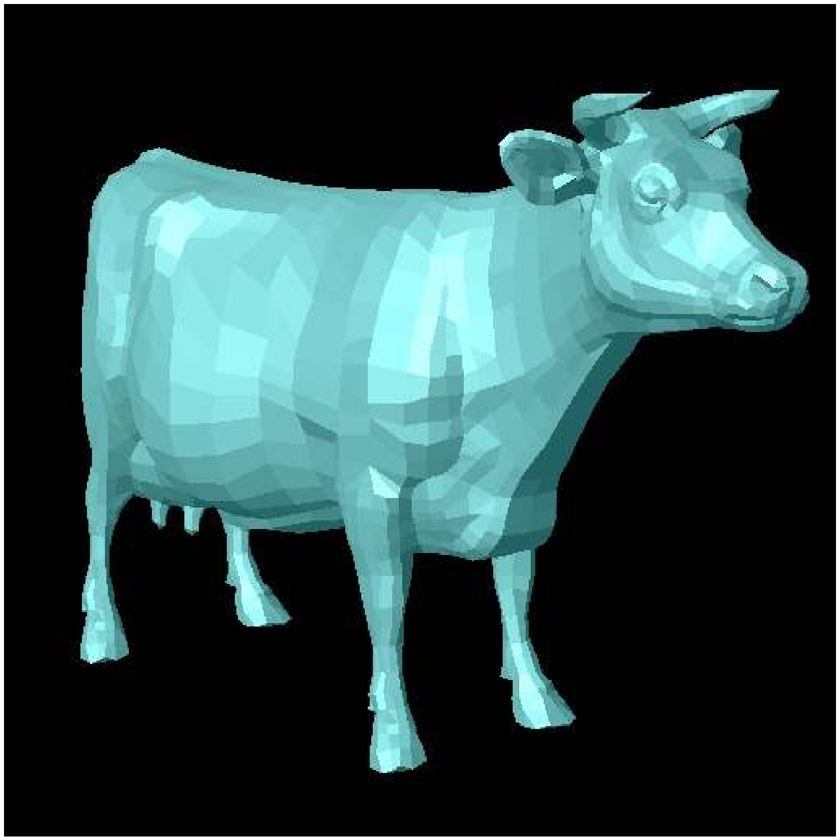
$\quad \forall$ verts ℓ in face k

$\qquad (n_{avg})_\ell \mathrel{+}= \hat{n}_k$

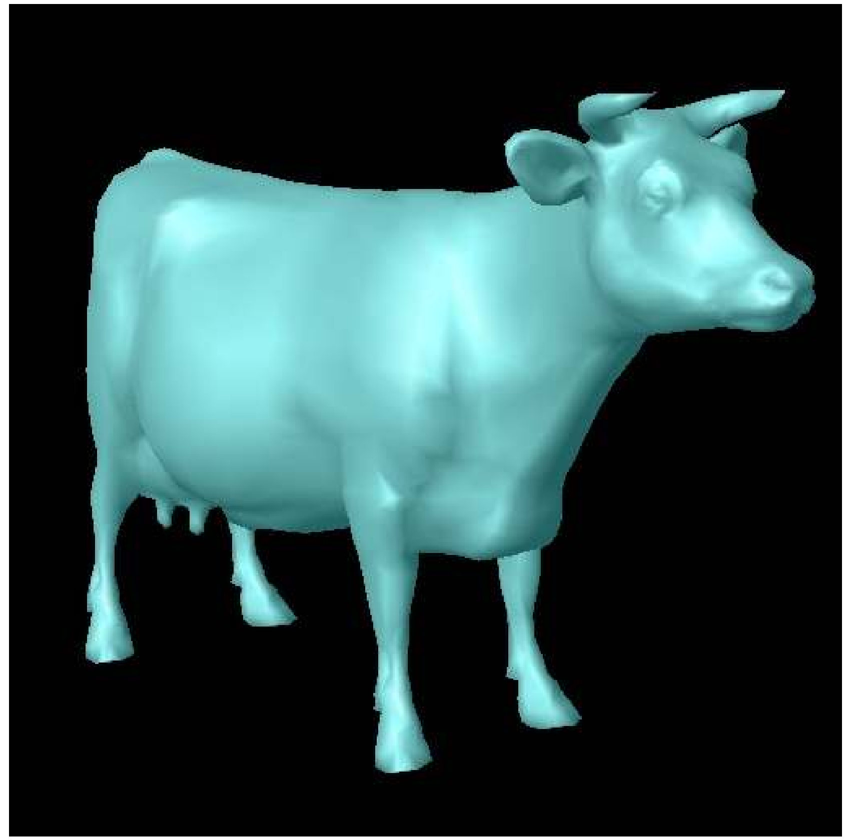
$\forall$ nodes j

$\quad$ normalize $(\hat{n}_{avg})_j$

Mesh rendering styles

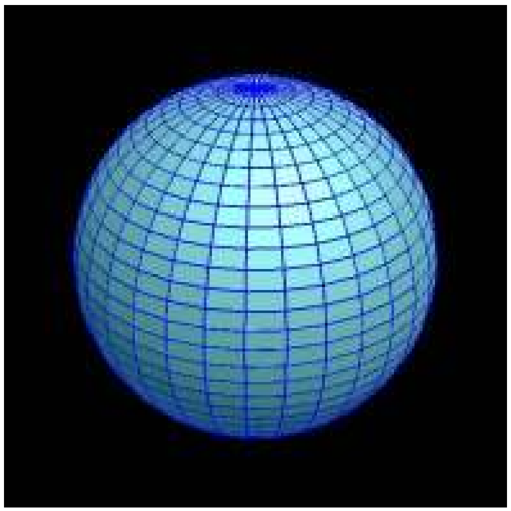


Flat (faceted) shading

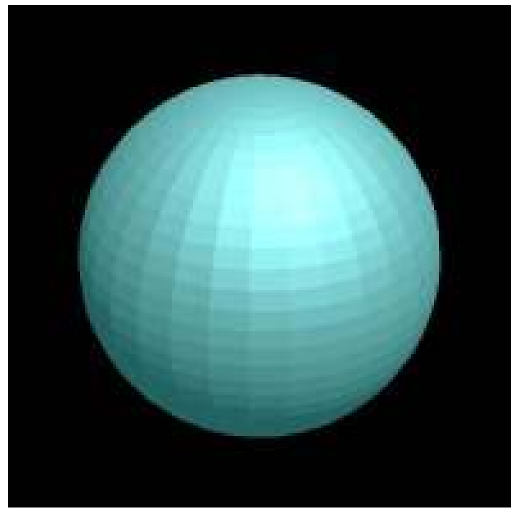


Smooth (Gouraud) shading

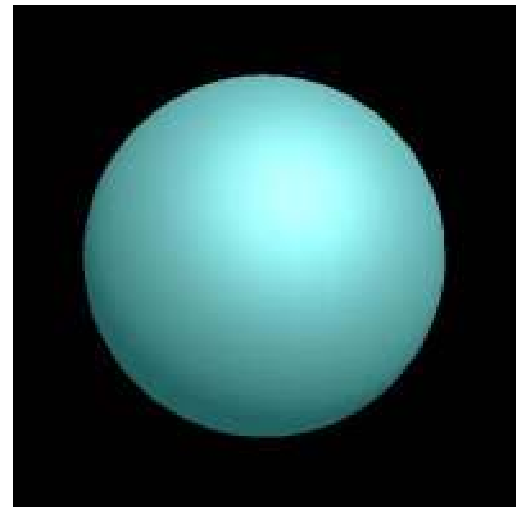
Sphere



Polygons and wireframe



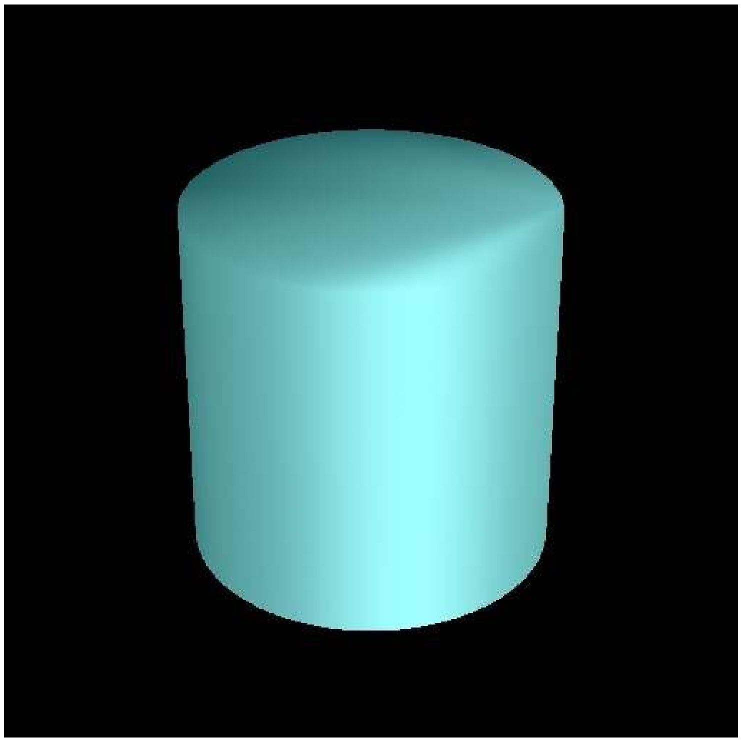
Flat



Smooth

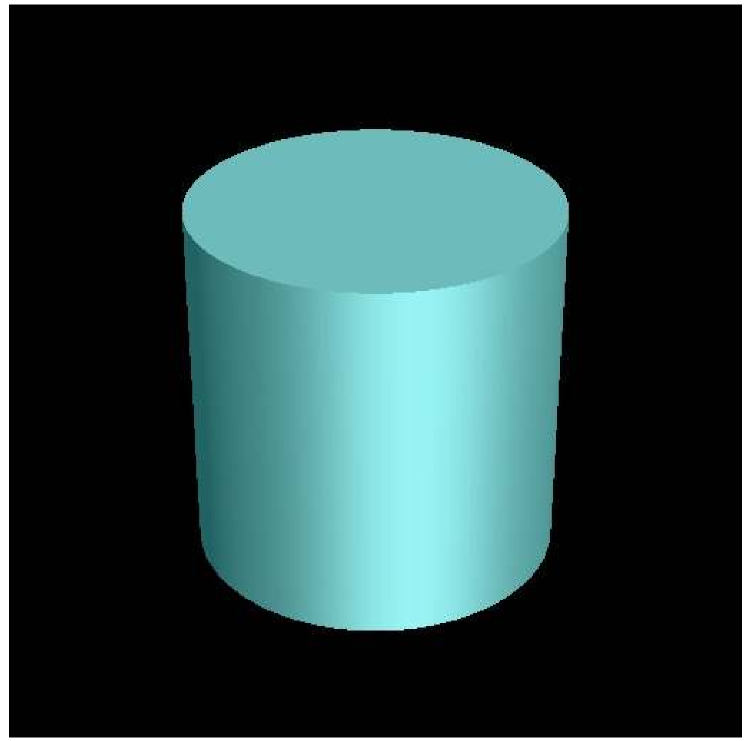
Vertex normals and smooth shading

Creases lost



Normal stored in vertex

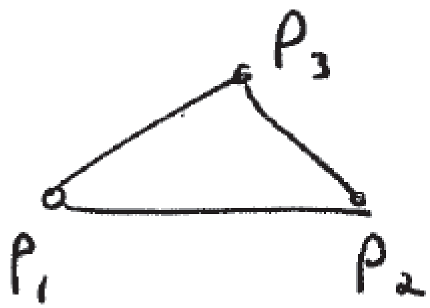
Creases retained



Normals stored in polygon (per vertex)

Polygon/surface orientation

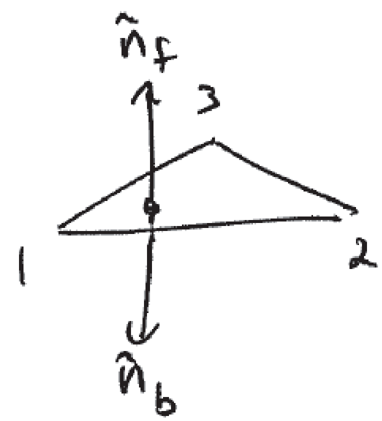
- Order of vertices specifies a polygon
- Backwards and forwards = same polygon



$$f = \{(1,2,3), (2,3,1), (3,1,2)\}$$

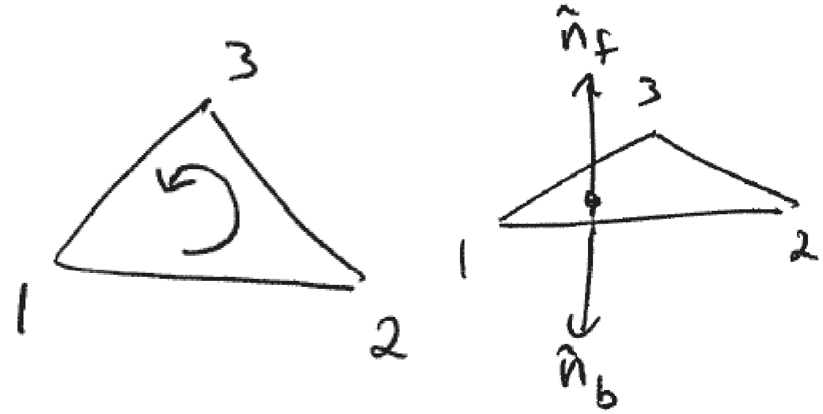
$$b = \{(3,2,1), (2,1,3), (1,3,2)\}$$

- But the normal direction flips



Polygon/surface orientation

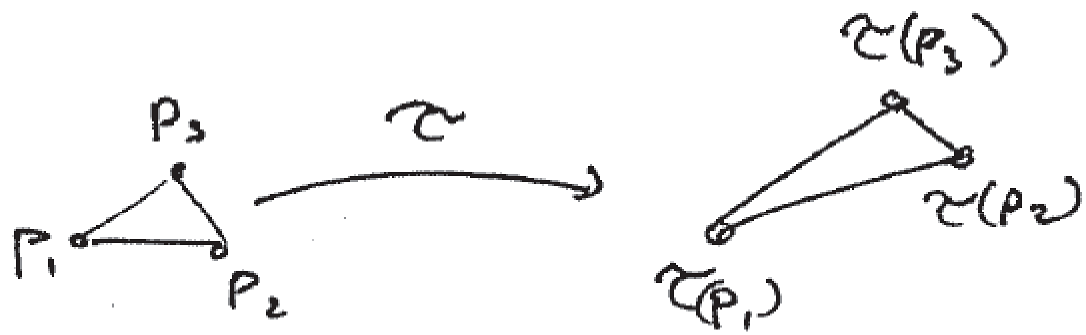
- Use right-hand rule to determine normal direction
  - Counter clockwise: normal comes out of “slide”
  - Convention: list vertices in CCW order



- Mesh should be consistently oriented
  - All point out!

Polygon transformation

- Transform points



- Draw polygon using these
    - Affine transformations map lines to lines (planes to planes, etc.)