

Computer Graphics

Raster Graphics

Subodh Kumar, Dept of Computer Sc. & Engg., IIT Delhi

Raster Graphics



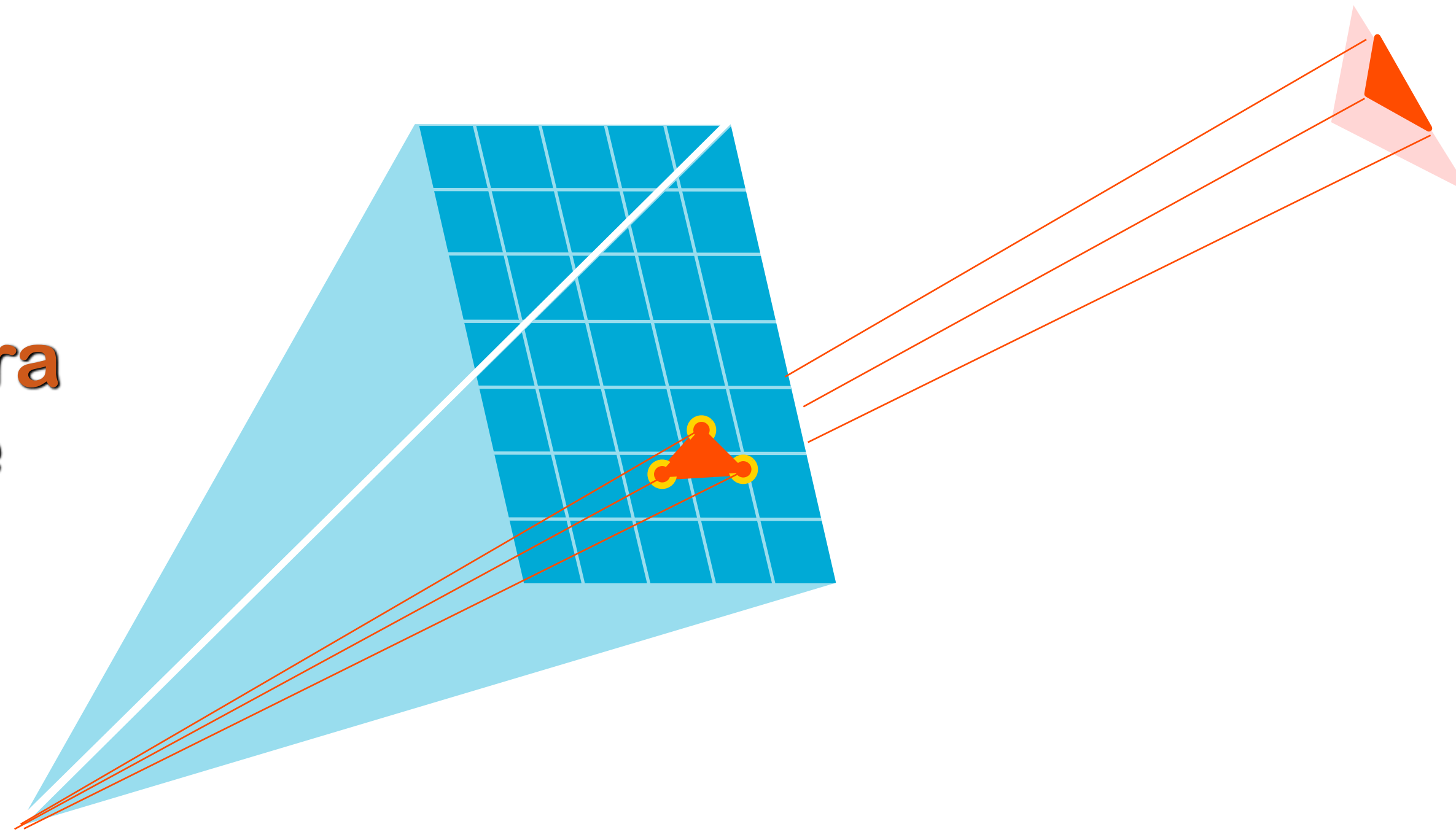
Project to 2D



- For each object, show it on screen



**Camera
Space**



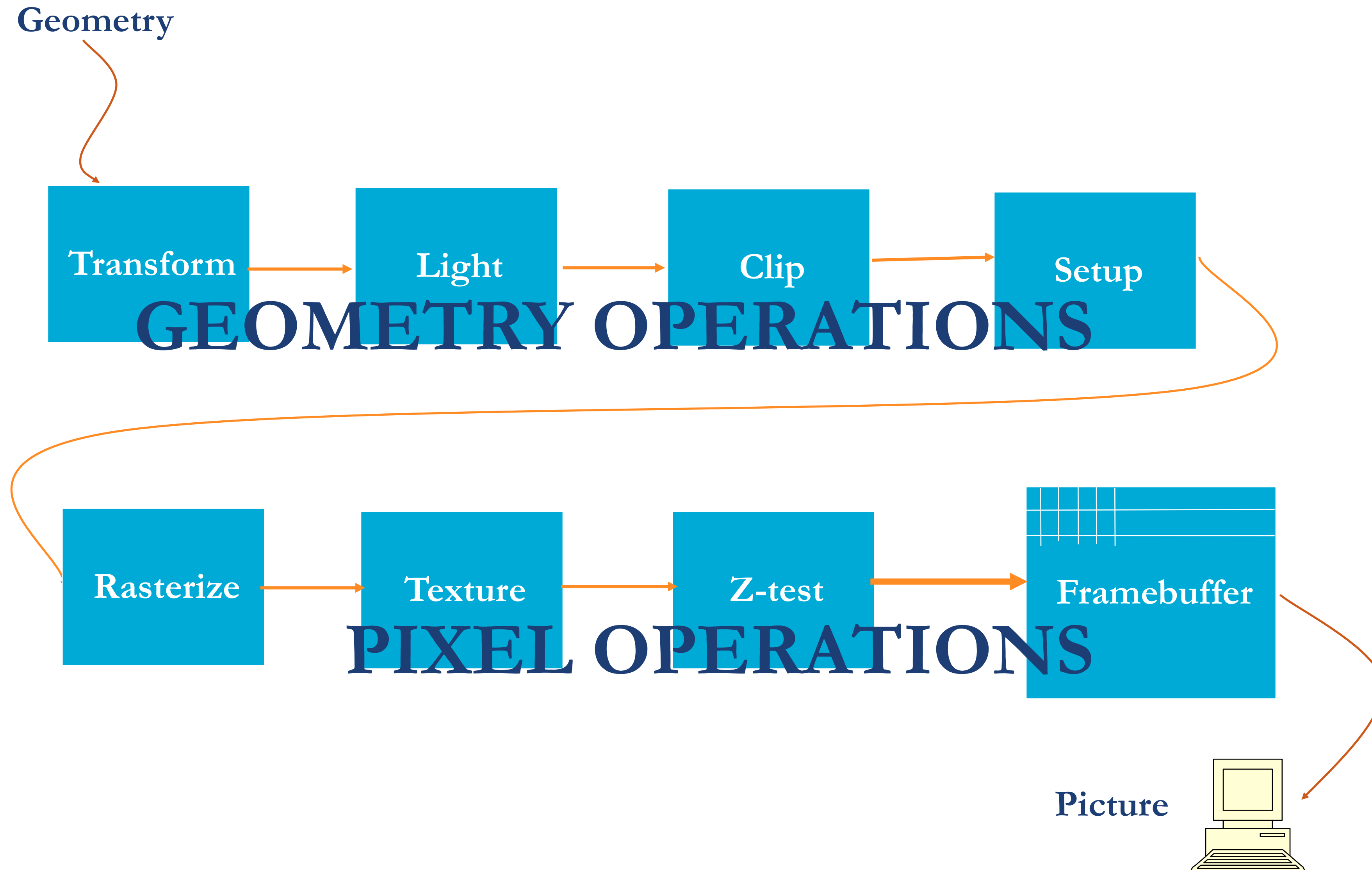
Raster Graphics



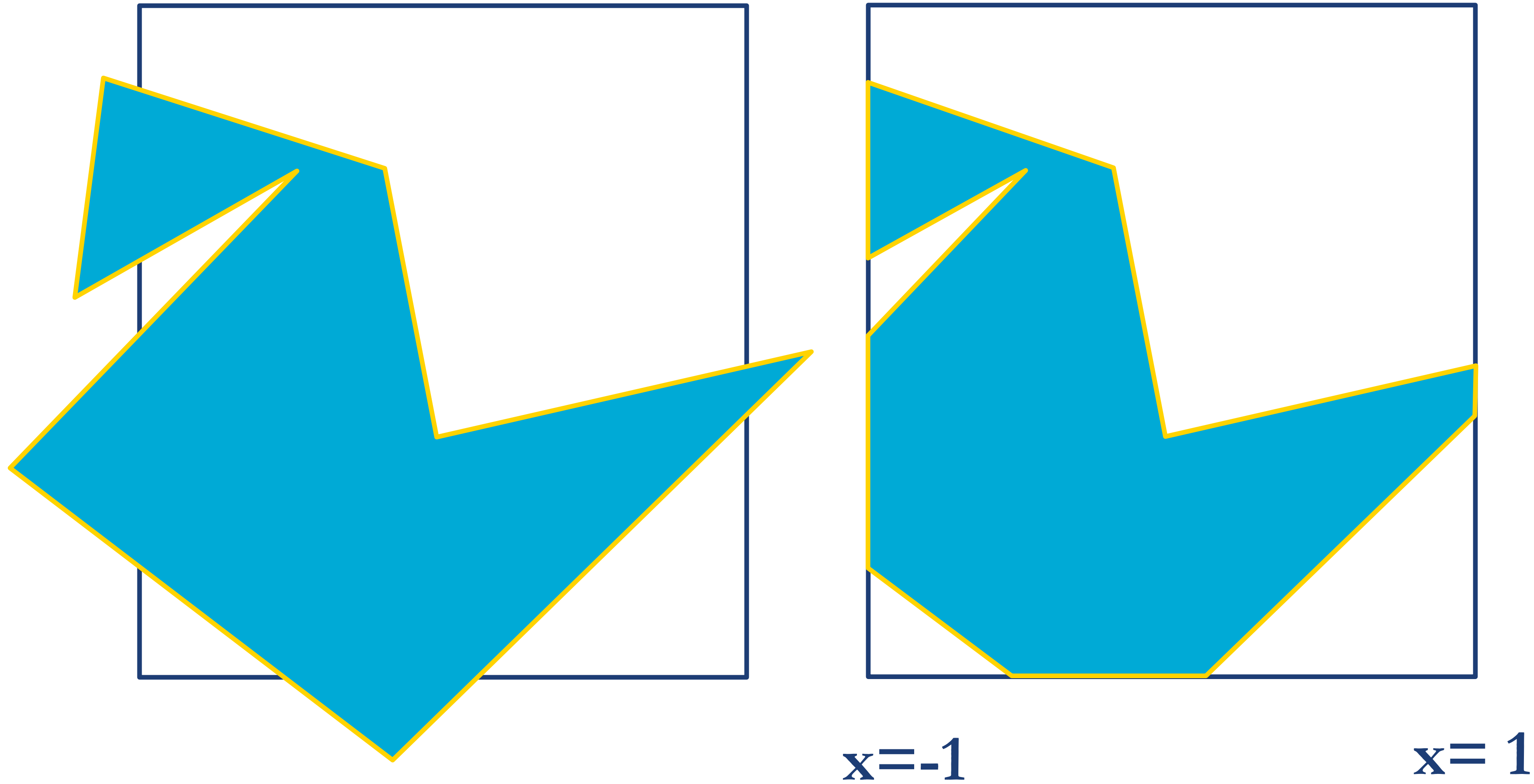
- For each triangle (or other geometry)
 - ➡ Transform and project vertices into camera frame
 - ➡ Find color at vertices
 - ➡ Fill the triangle formed by those vertices with interpolated colors
 - Texture look-up for more detail
 - ➡ For each resulting pixel
 - Store if the closest found so far



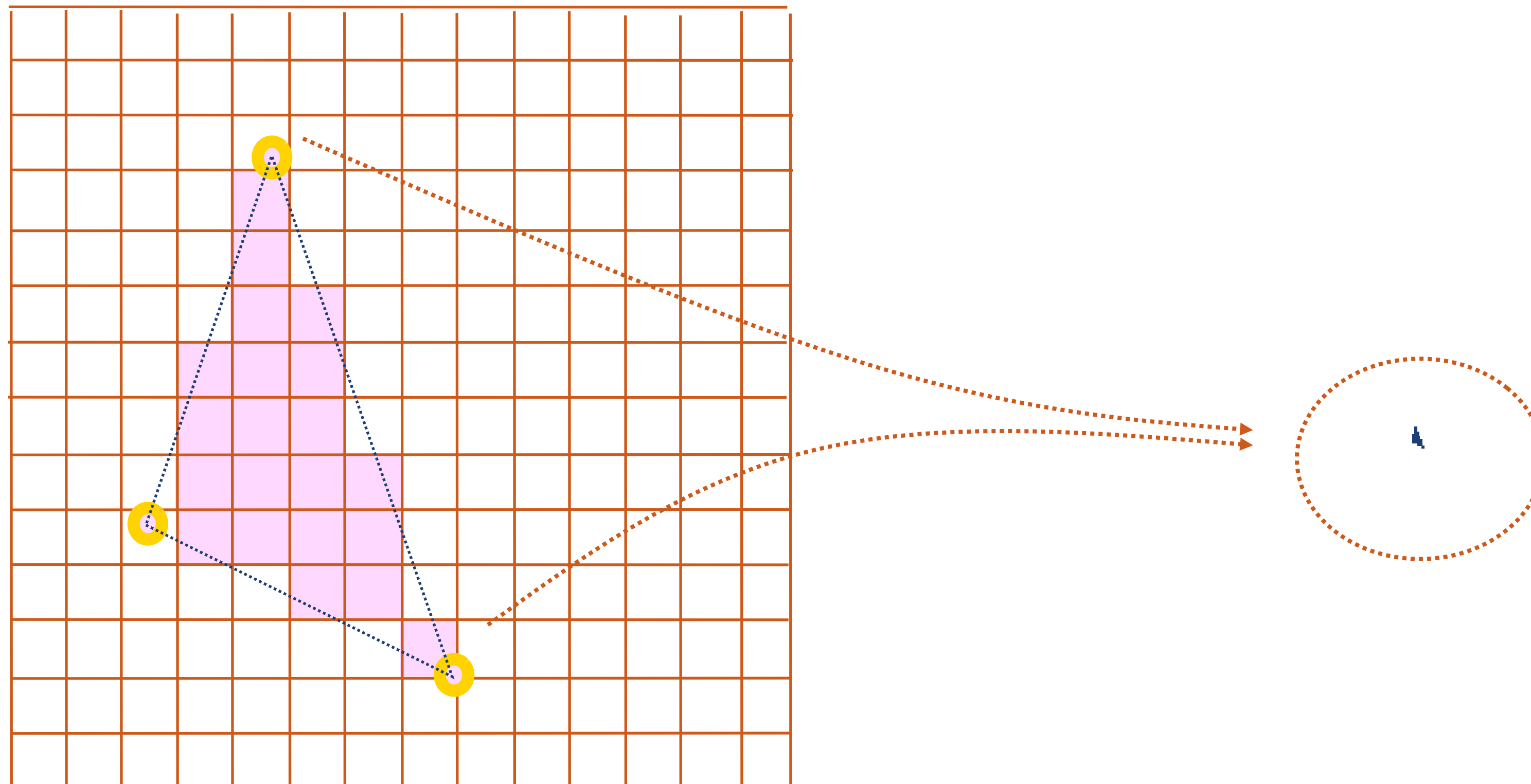
Rendering Pipeline



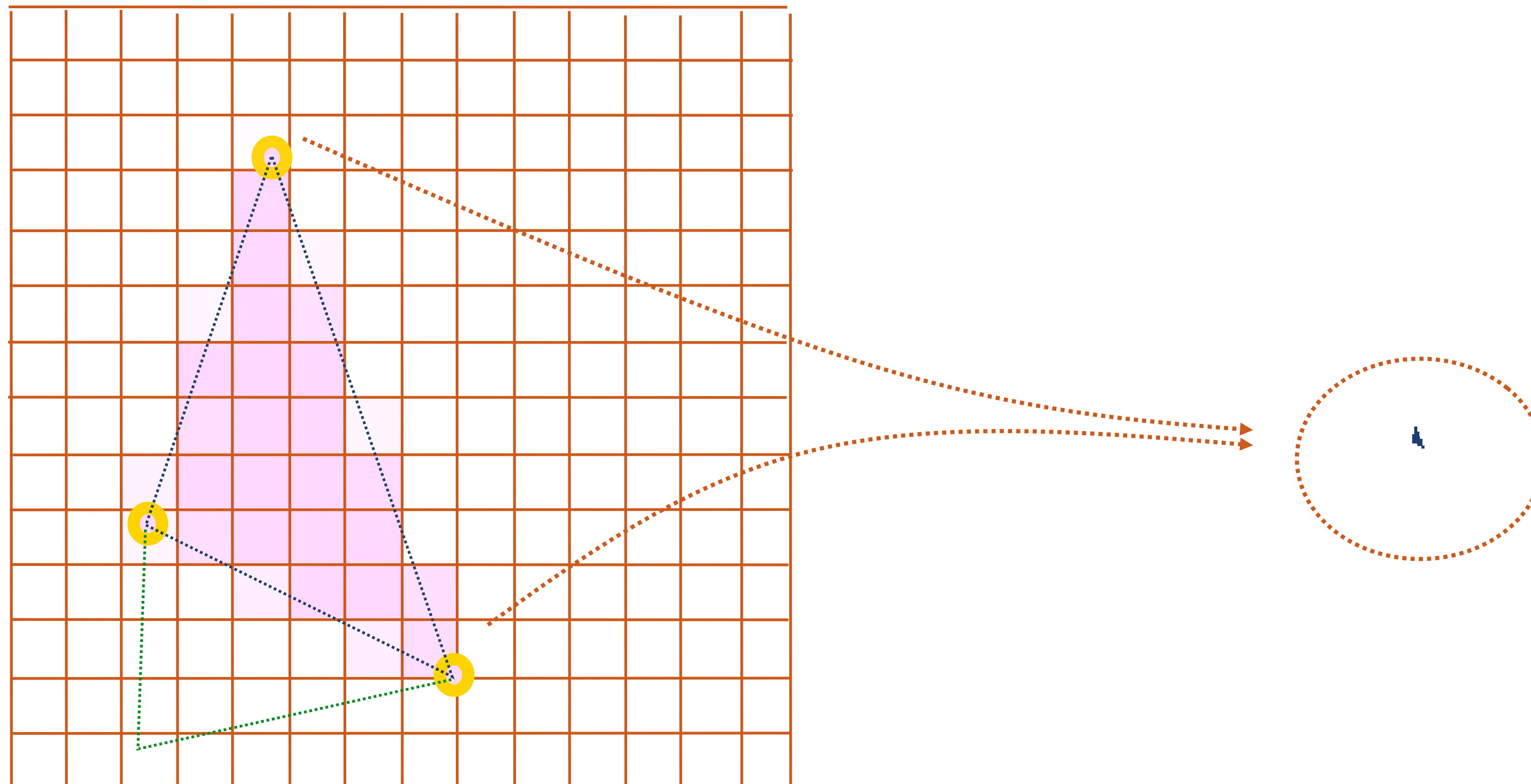
Clipping



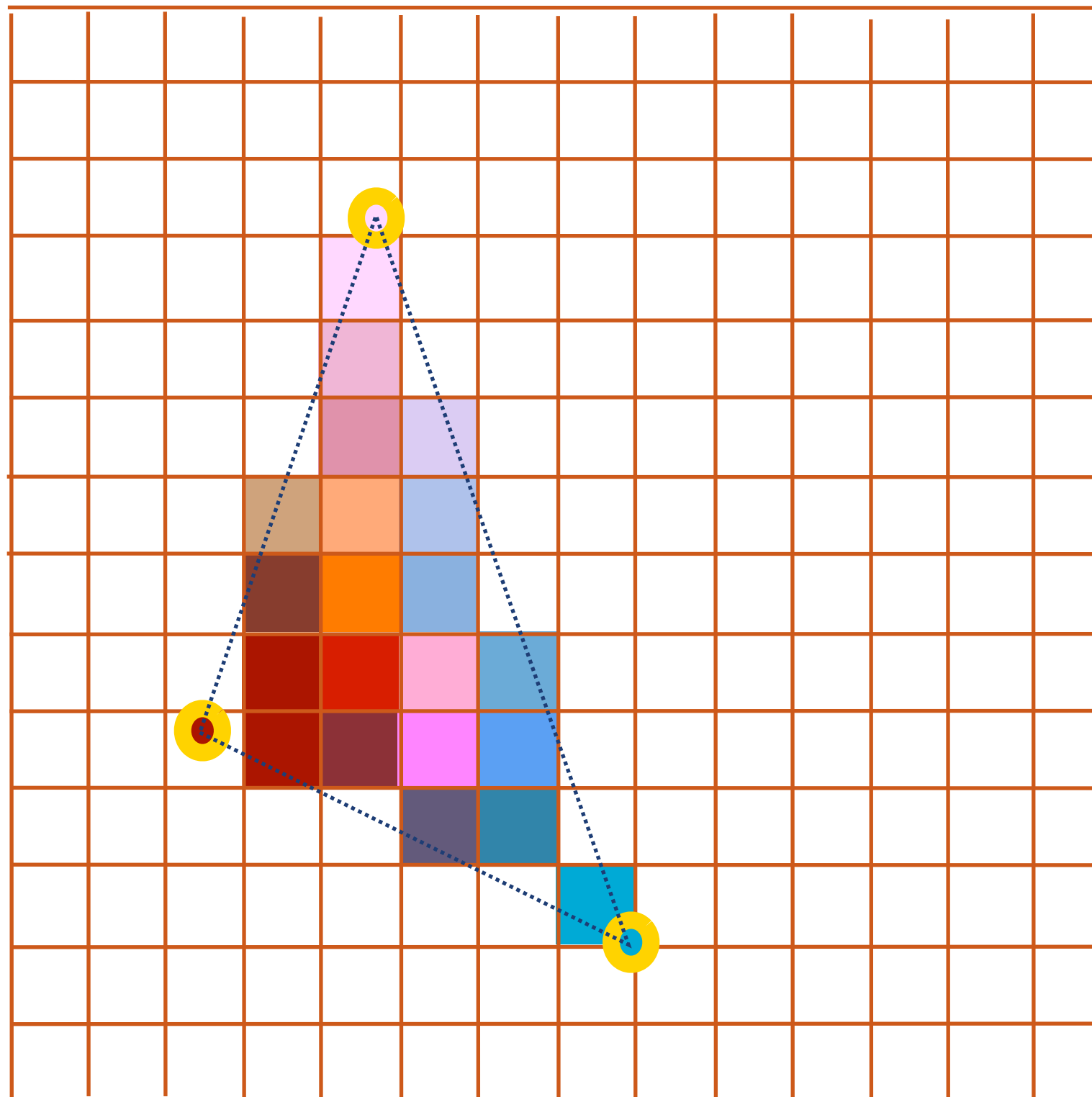
Rasterization: Fill



Rasterization: Anti-alias



Rasterization: Shade





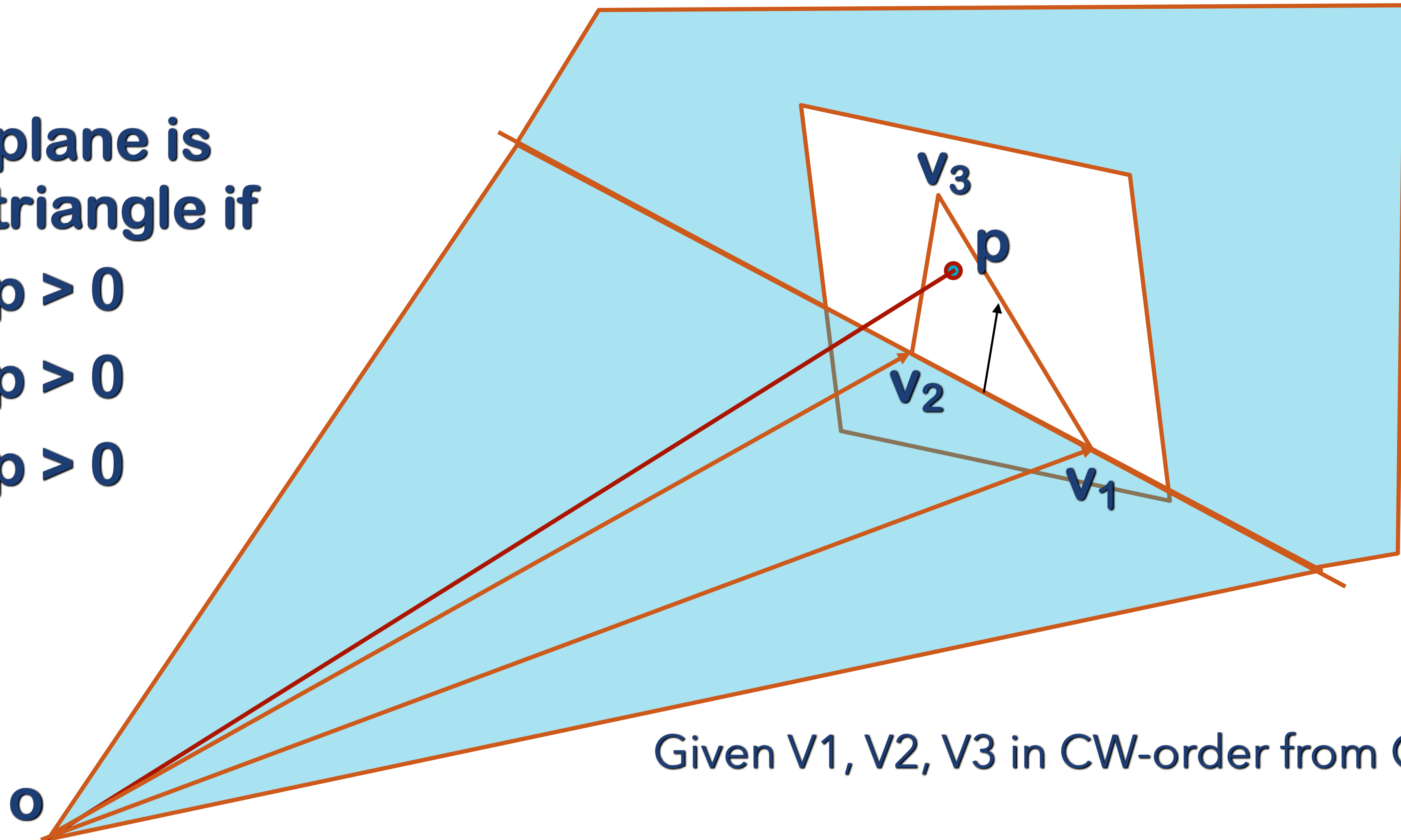
Half-plane Method

point p on plane is
inside the triangle if

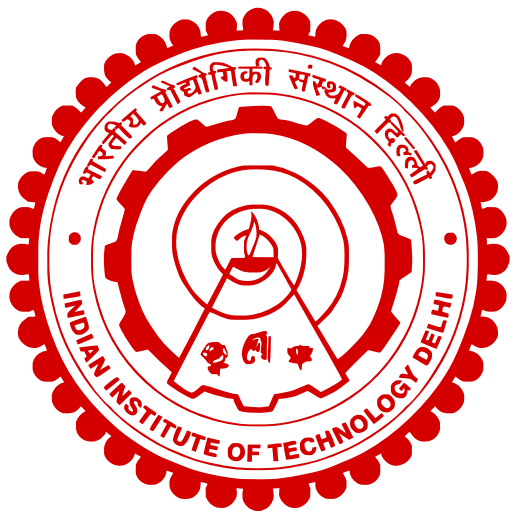
$$(v_1 \times v_2) \cdot p > 0$$

$$(v_2 \times v_3) \cdot p > 0$$

$$(v_3 \times v_1) \cdot p > 0$$



Given V_1, V_2, V_3 in CW-order from Origin

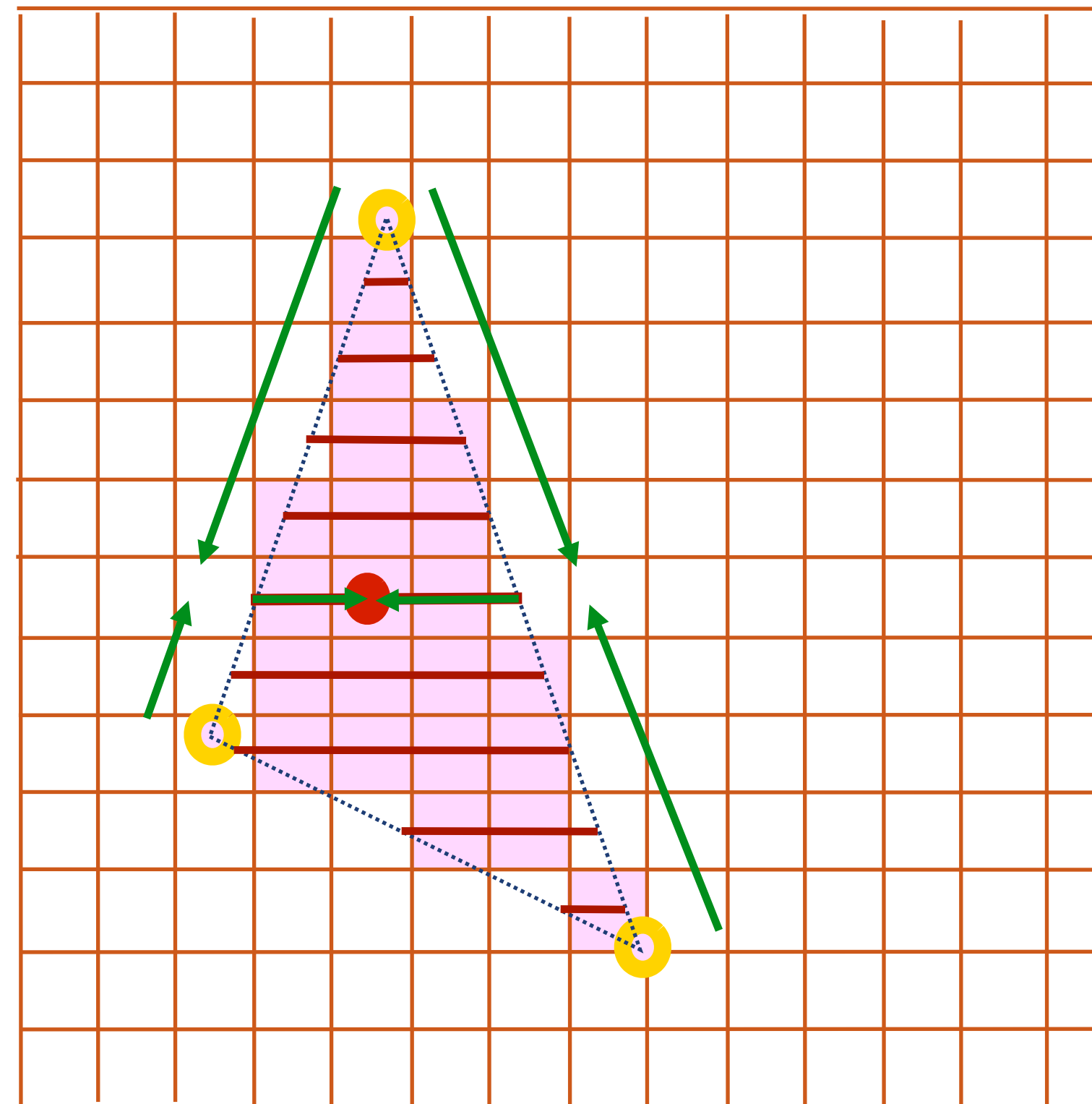


Raster Graphics Steps

- Locate the projection of 3D vertices in 2D 'screen'
- Fill: Determine the samples which lie 'within' the extents of primitive
- Position each sample with respect to the primitive
- Finally compute the color
 - ➔ May need to discard or combine it (with other primitives)

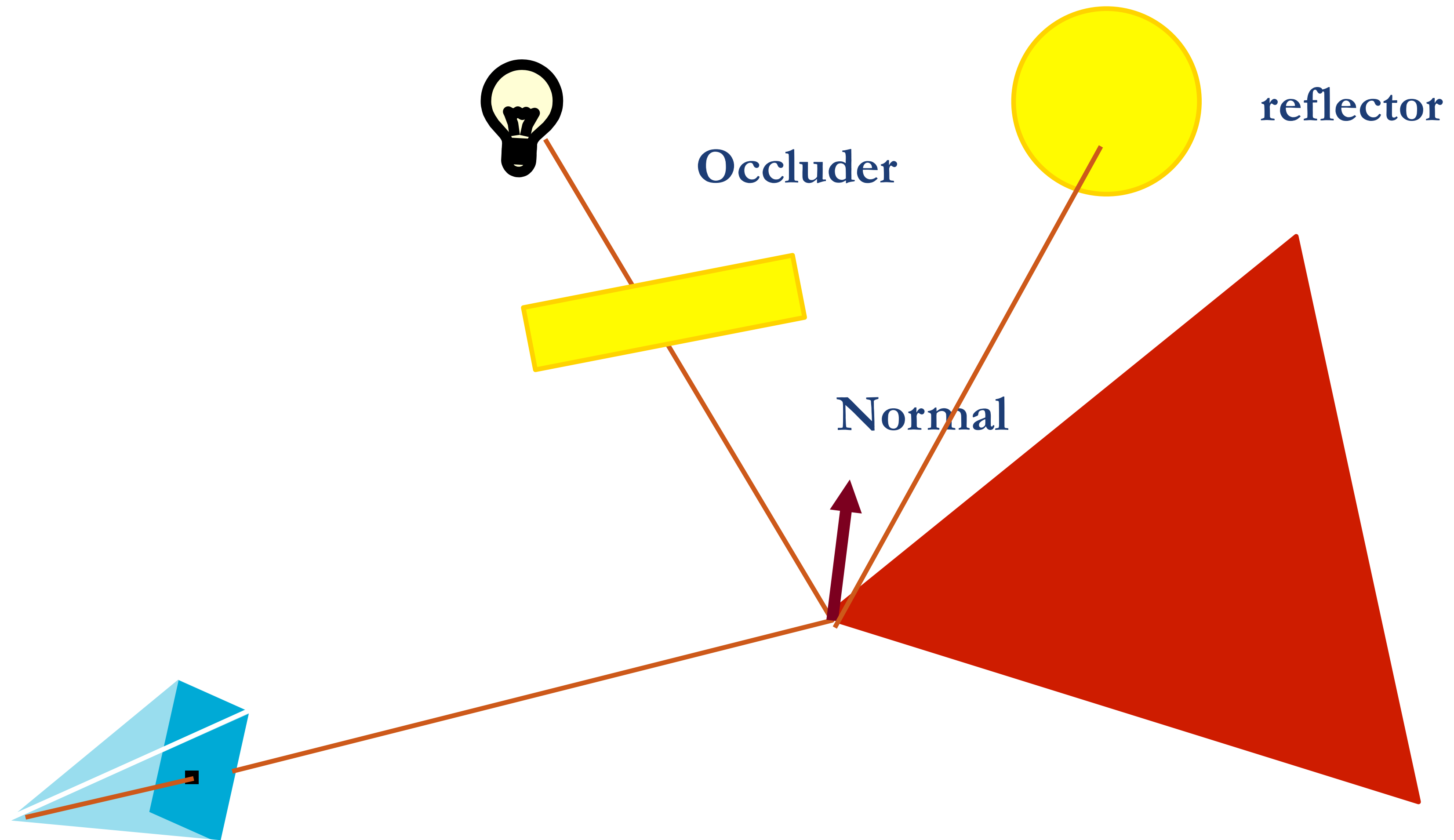
Rasterization: Process

- Find low **scanline**
- Find high scanline
- For each scanline:
 - ➡ Find Left End
 - ➡ Find Right End
 - ➡ Fill **span**
 - Interpolate **Attributes**
 - $a_0t + a_1(1-t)$





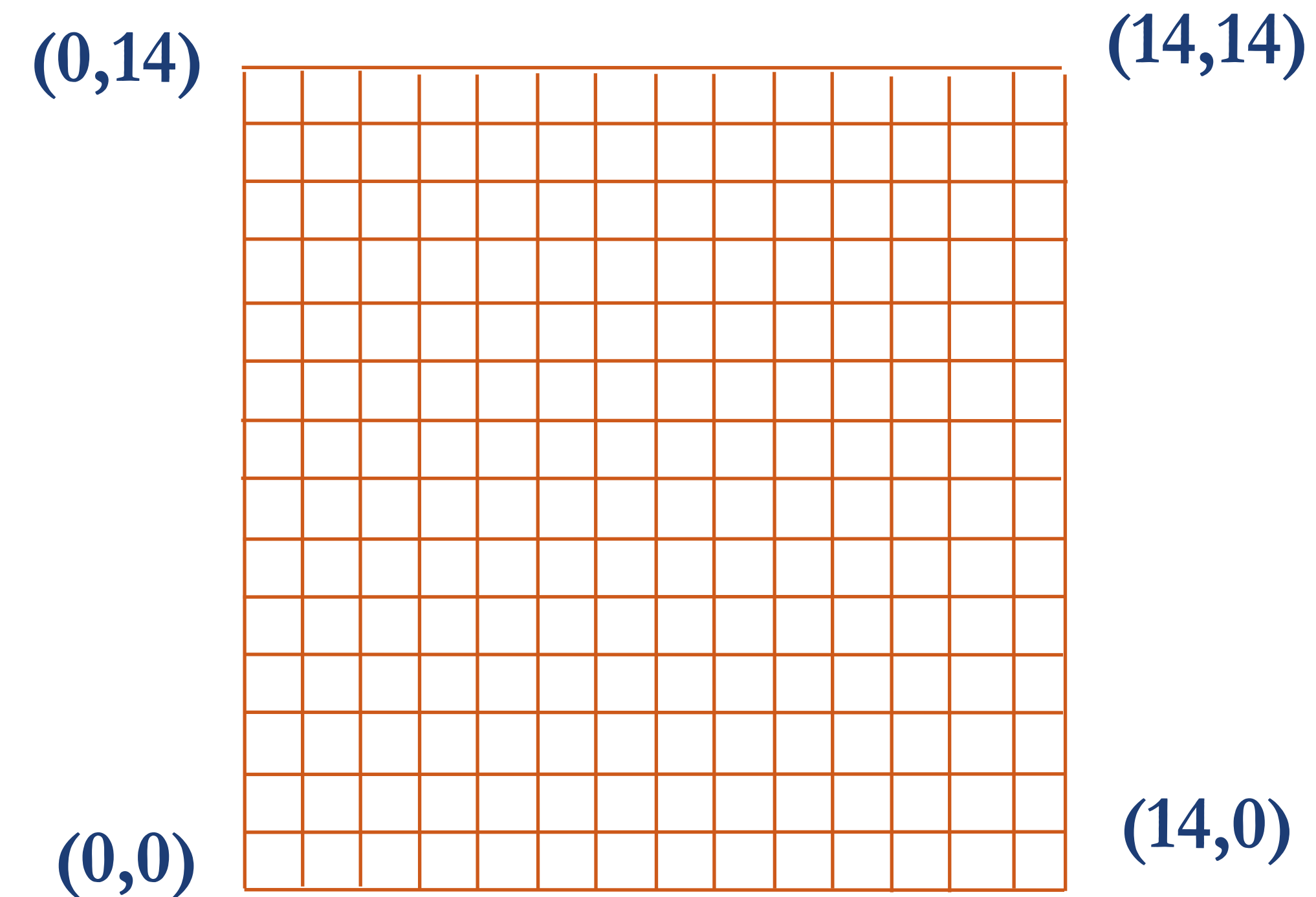
Lighting





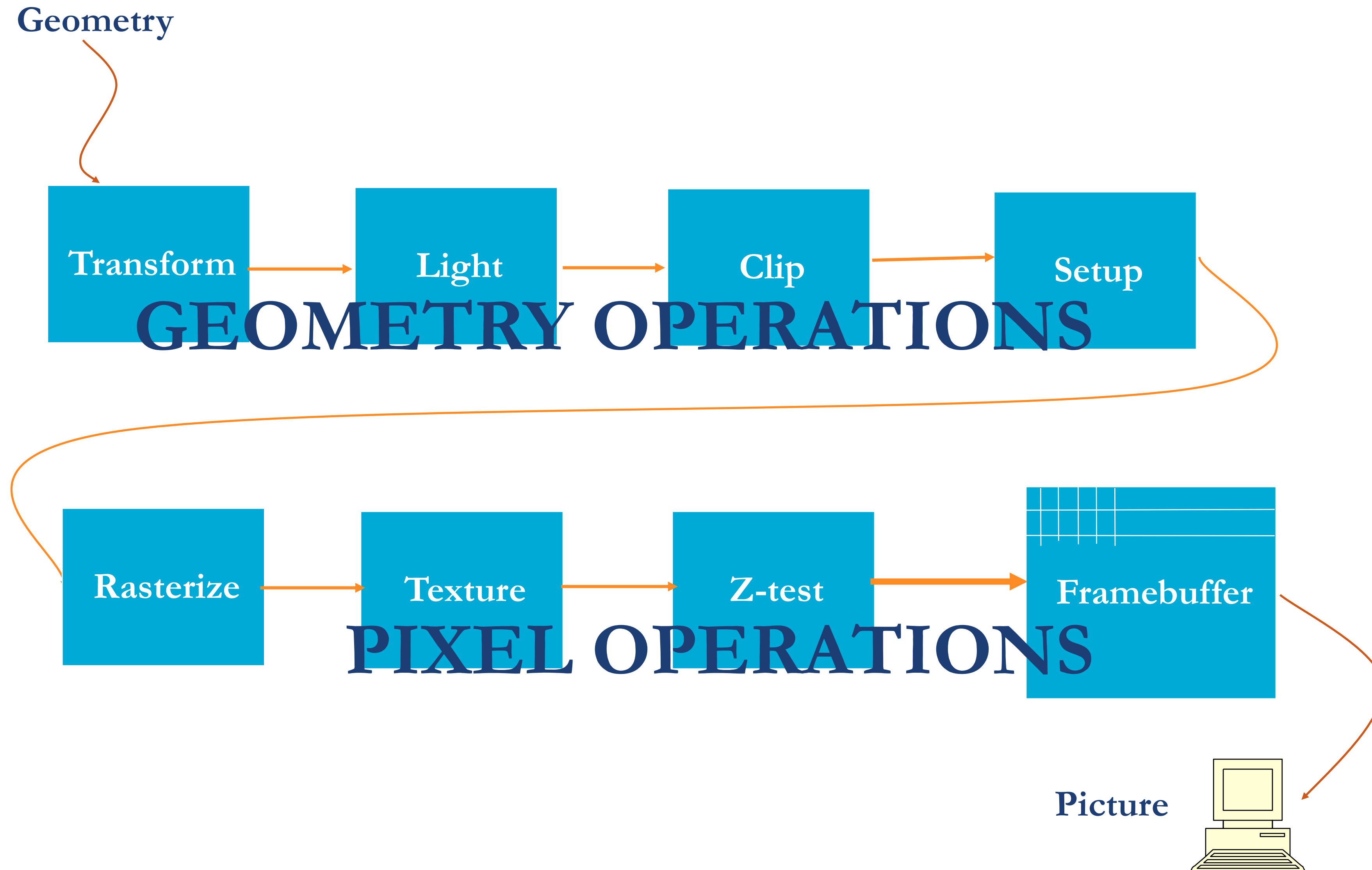
Transformation

- Change to a common coordinate system
- Change to a convenient coordinate system, eg,





Rendering Pipeline

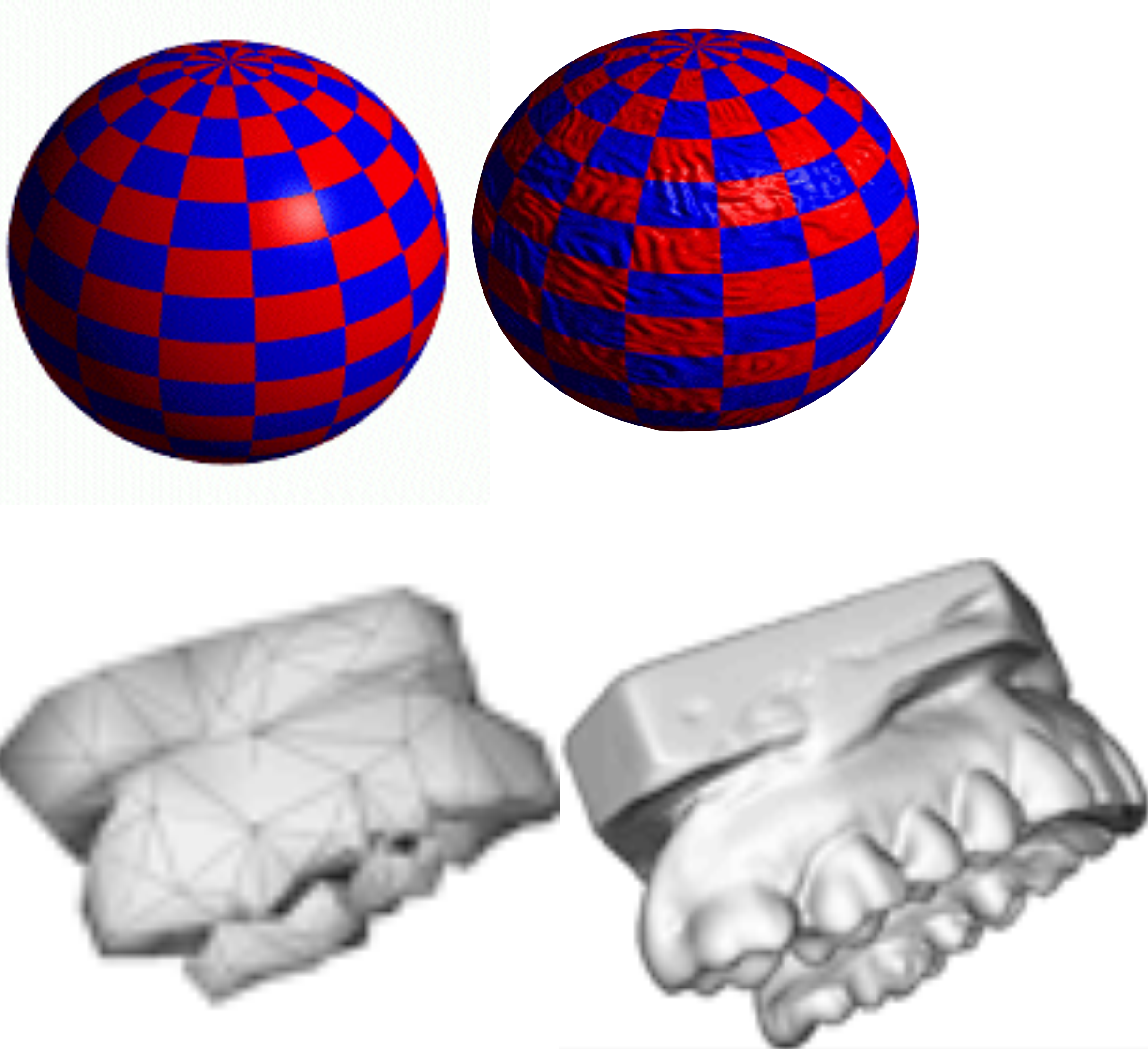




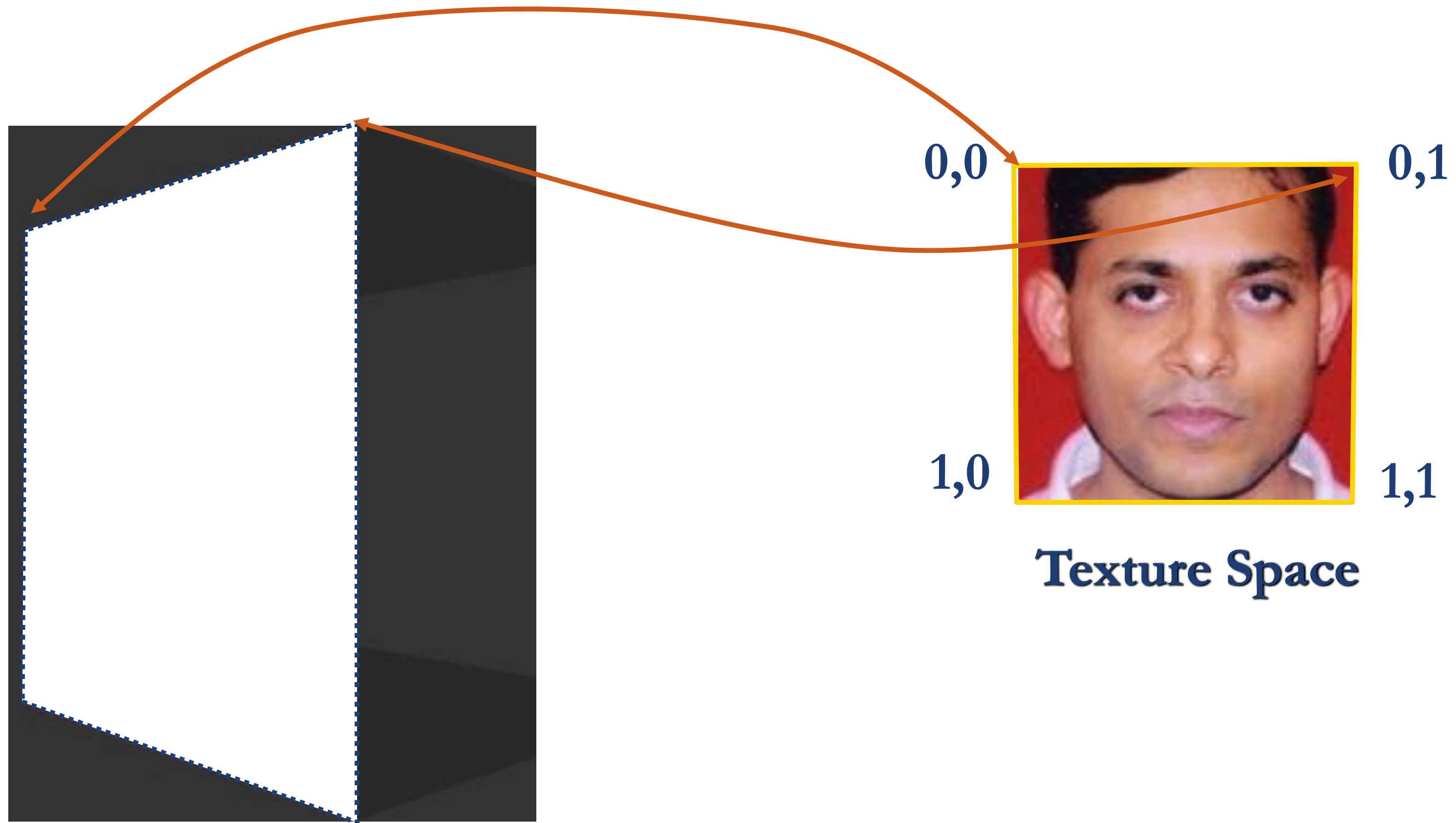
Are Triangles Sufficient?

- Curved shapes
 - ➡ Usually approximated with (many) triangles anyway
 - ➡ But vary color smoothly across a triangle
- A function
 - ➡ Eg, Noise (fractals)
- A cloud of "particles"
- 3D grid (volumes)
- Decorated polygons

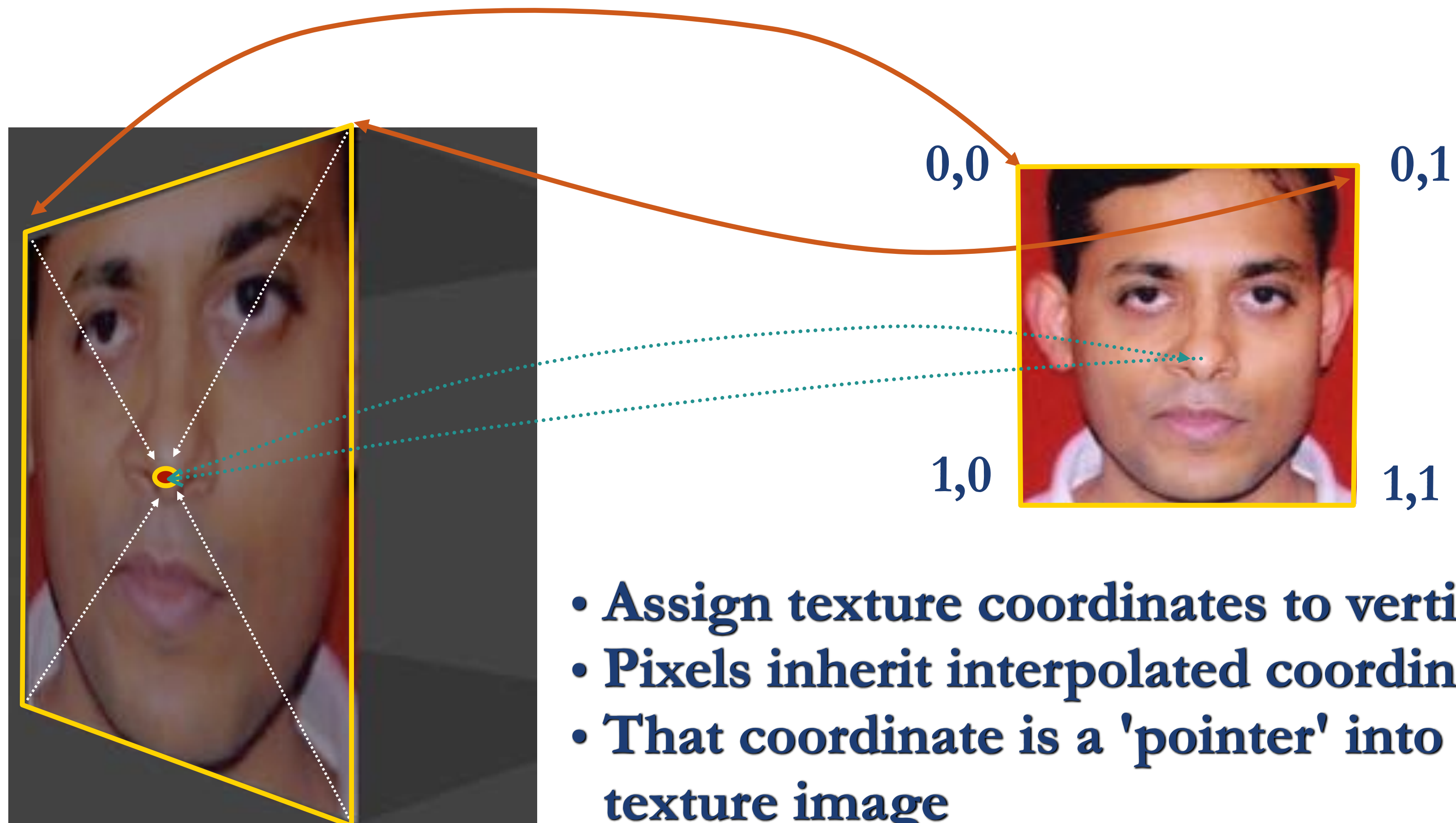
Unsmooth



Unsmooth with Texture Mapping

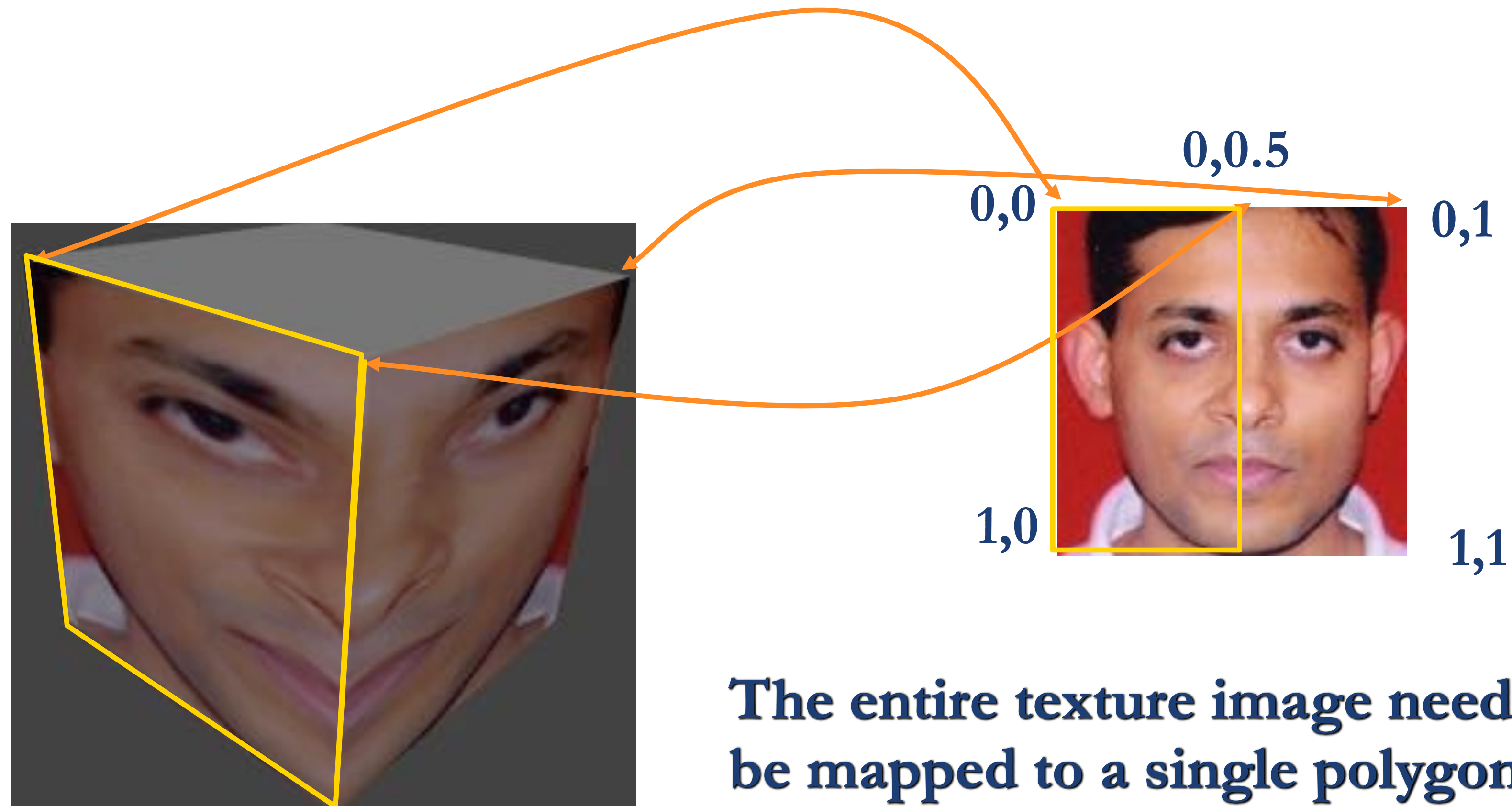


About Texture Mapping



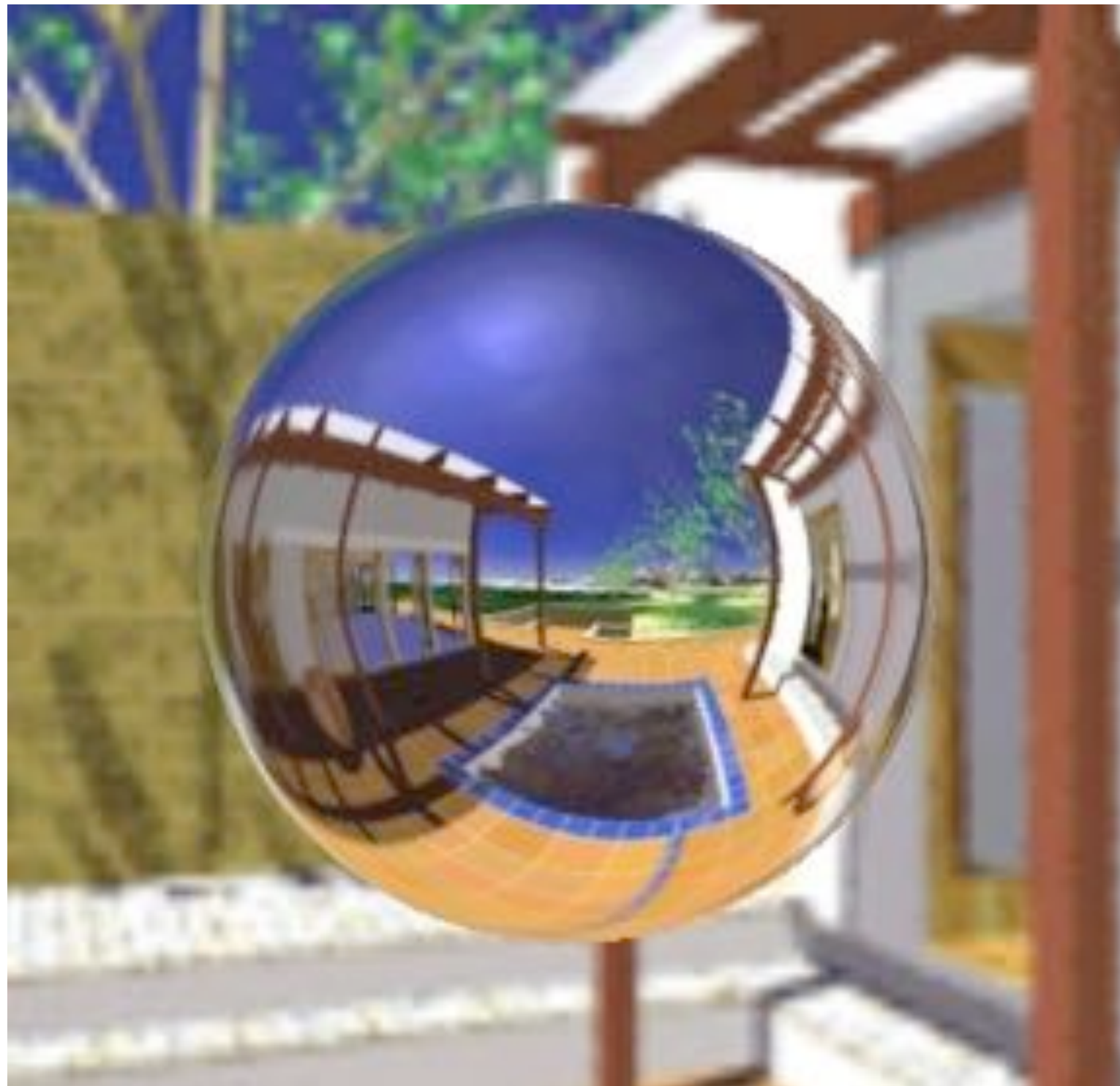
- Assign texture coordinates to vertices
- Pixels inherit interpolated coordinates
- That coordinate is a 'pointer' into the texture image

Texture Coordinates

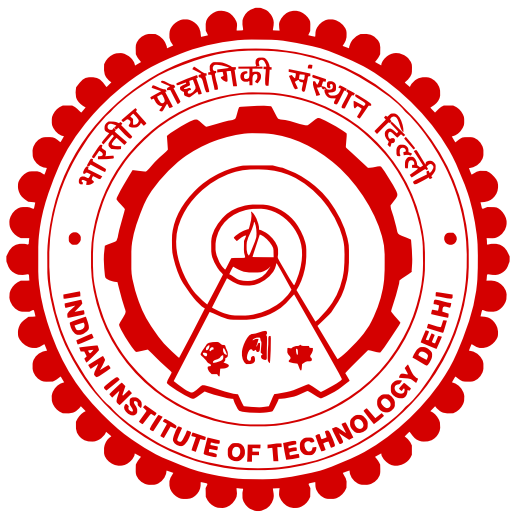


The entire texture image need not be mapped to a single polygon

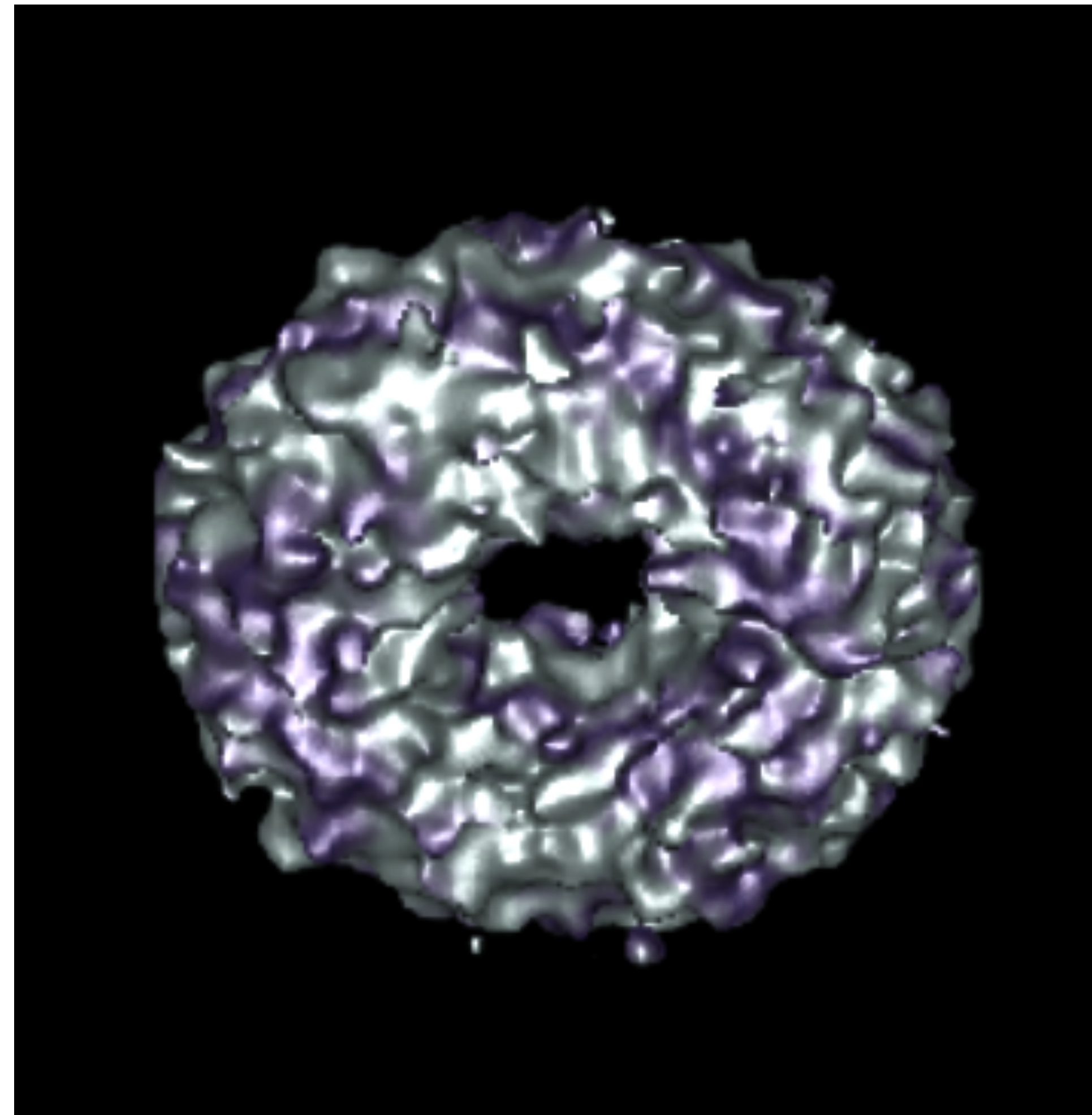
Environment Mapping



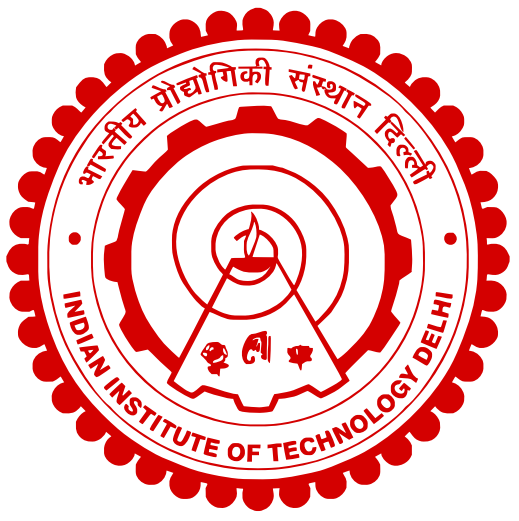
Bump Mapping



Displacement Mapping



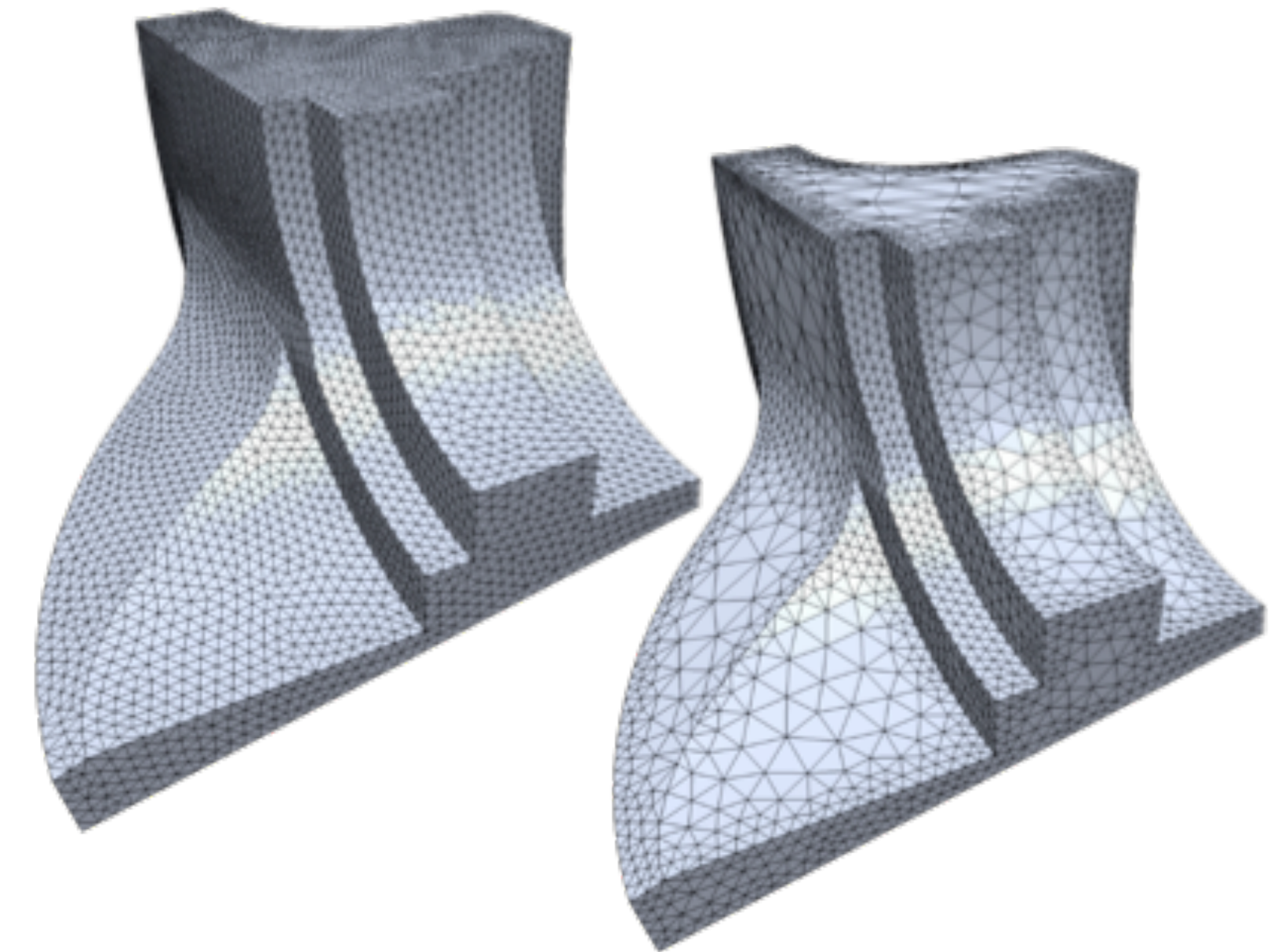
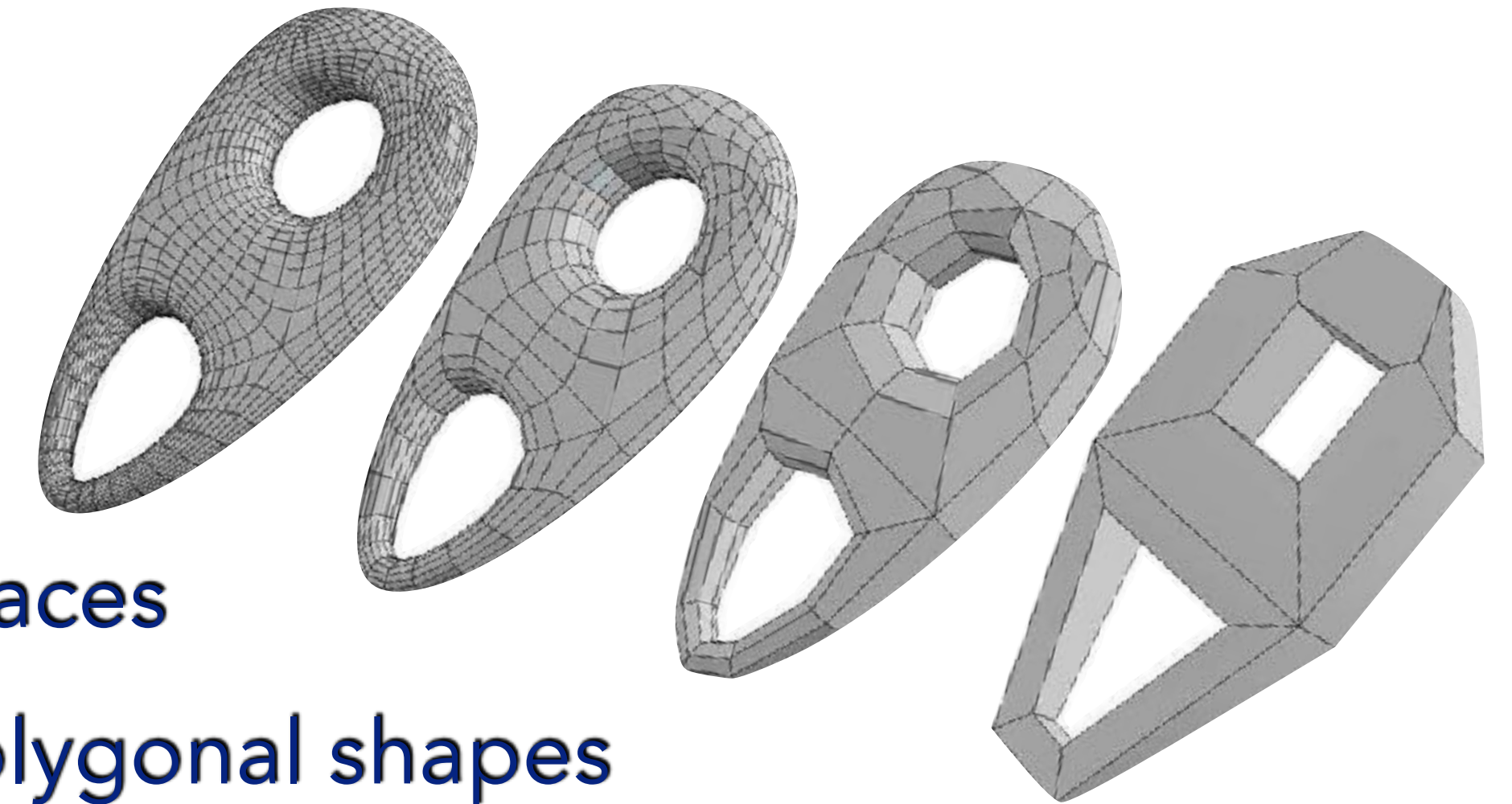
Surface as the Model of Objects



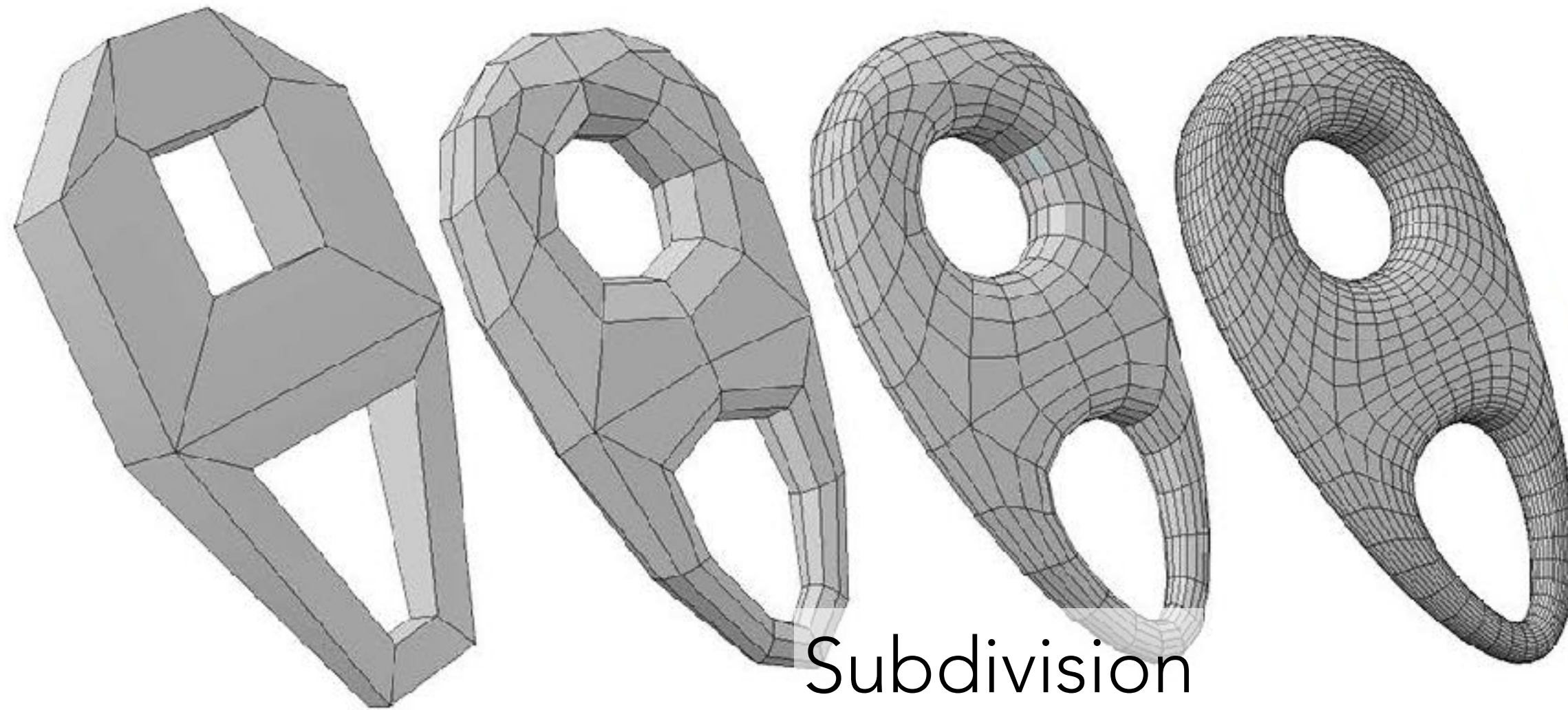
- We only see the apparent surface of most objects
- Surface can be sampled by many points
 - ➔ However surface is continuous and finite (usually)
 - ➔ Each point has 'zero' size
- Need a way to cover the entire surface: fill in the gaps
 - ➔ disk, rectangle, ellipse, blobs etc.
 - ➔ Also need the surface normals for lighting
- Easier if samples cover the surface and each surface point is sampled only once

Polygon meshes

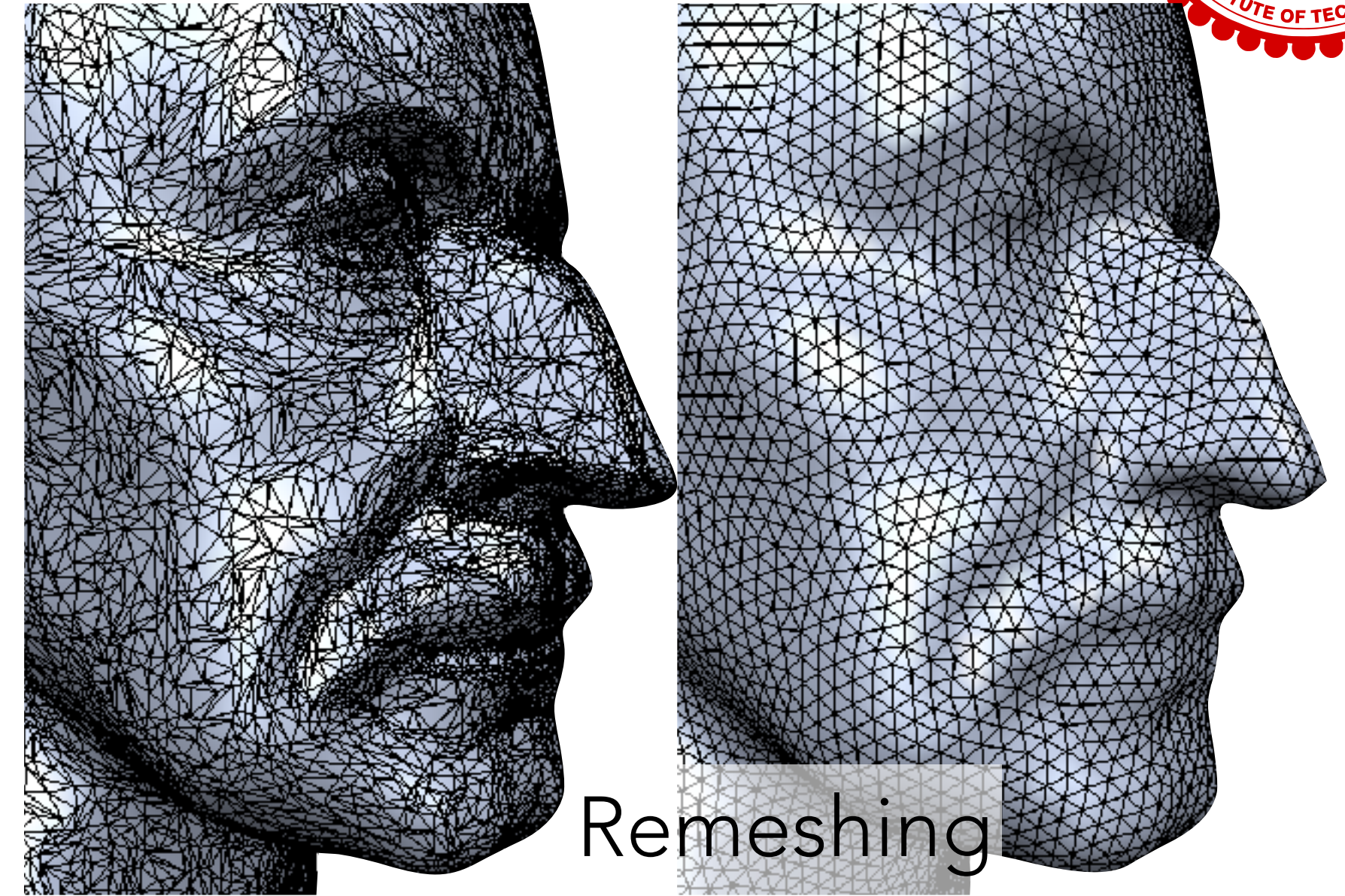
- Triangles are the simplest shapes that fit
 - ➔ Linear elements, efficient to render
 - ➔ Can control the closeness of triangles to curved surfaces
 - ➔ Sometimes, we use quadrilaterals or other planar polygonal shapes
- Do not overlap and do not leave gaps
 - ➔ Share edges (and vertices)
 - ➔ Triangles share edges with with three other triangles
- Vertices may also carry surface normals
 - ➔ allows smooth interpolation of normals within the triangle



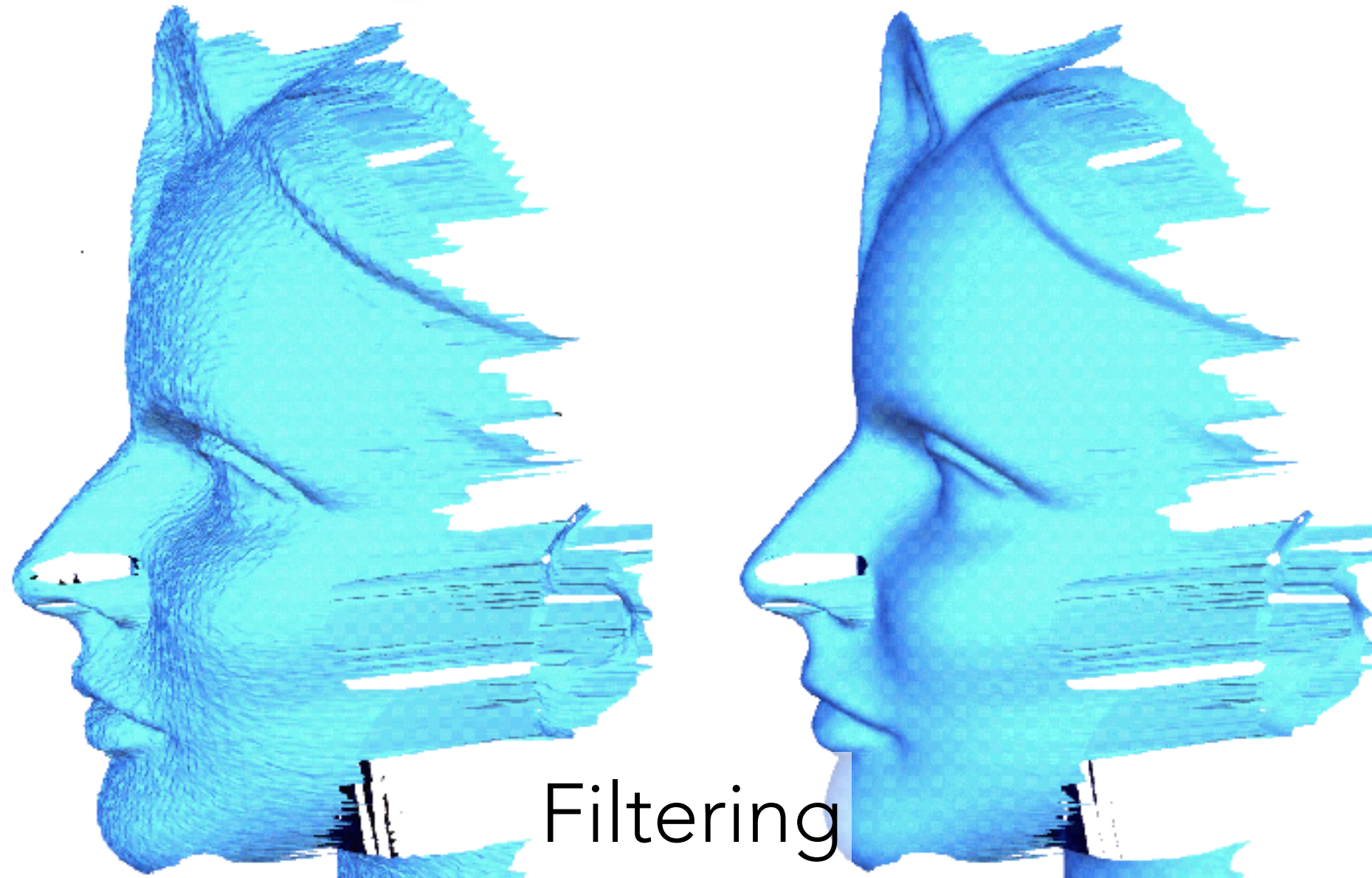
Polygon Mesh Operations



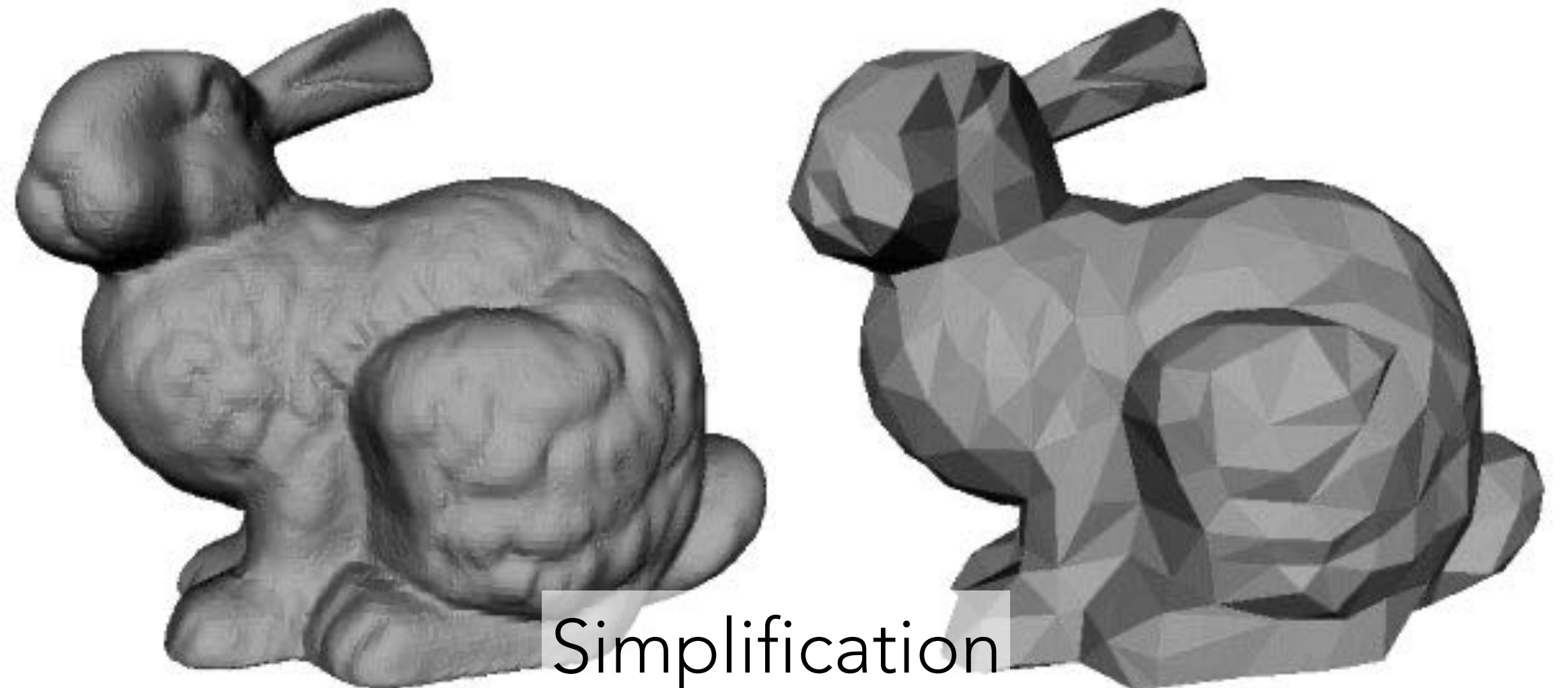
Subdivision



Remeshing



Filtering



Simplification



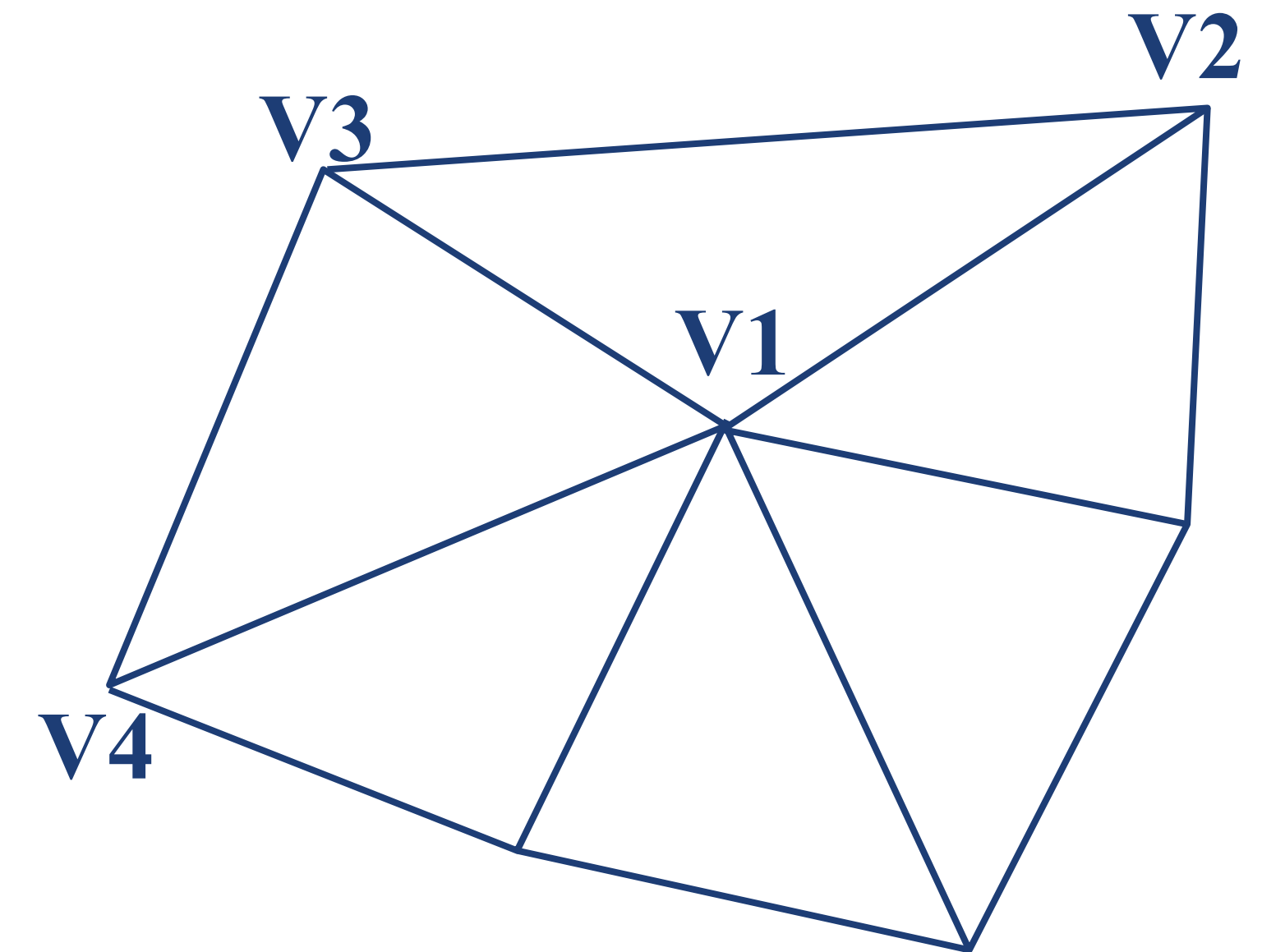
Triangle Representation

- Polygon soup

- ➡ Redundant storage of vertices (geometry)
- ➡ No connectivity information (topology)

$V1_x$	$V1_y$	$V1_z$	$V2_x$	$V2_y$	$V2_z$	$V3_x$	$V3_y$	$V3_z$
$V1_x$	$V1_y$	$V1_z$	$V3_x$	$V3_y$	$V3_z$	$V4_x$	$V4_y$	$V4_z$
...								

$V1_x$	$V1_y$	$V1_z$	$n1_x$	$n1_y$	$n1_z$	$V2_x$	$V2_y$	$V2_z$	$n2_x$	$n2_y$	$n2_z$	$V3_x$	$V3_y$	$V3_z$	$n3_x$	$n3_y$	$n3_z$
...																	



Other attributes: e.g., texture coordinates

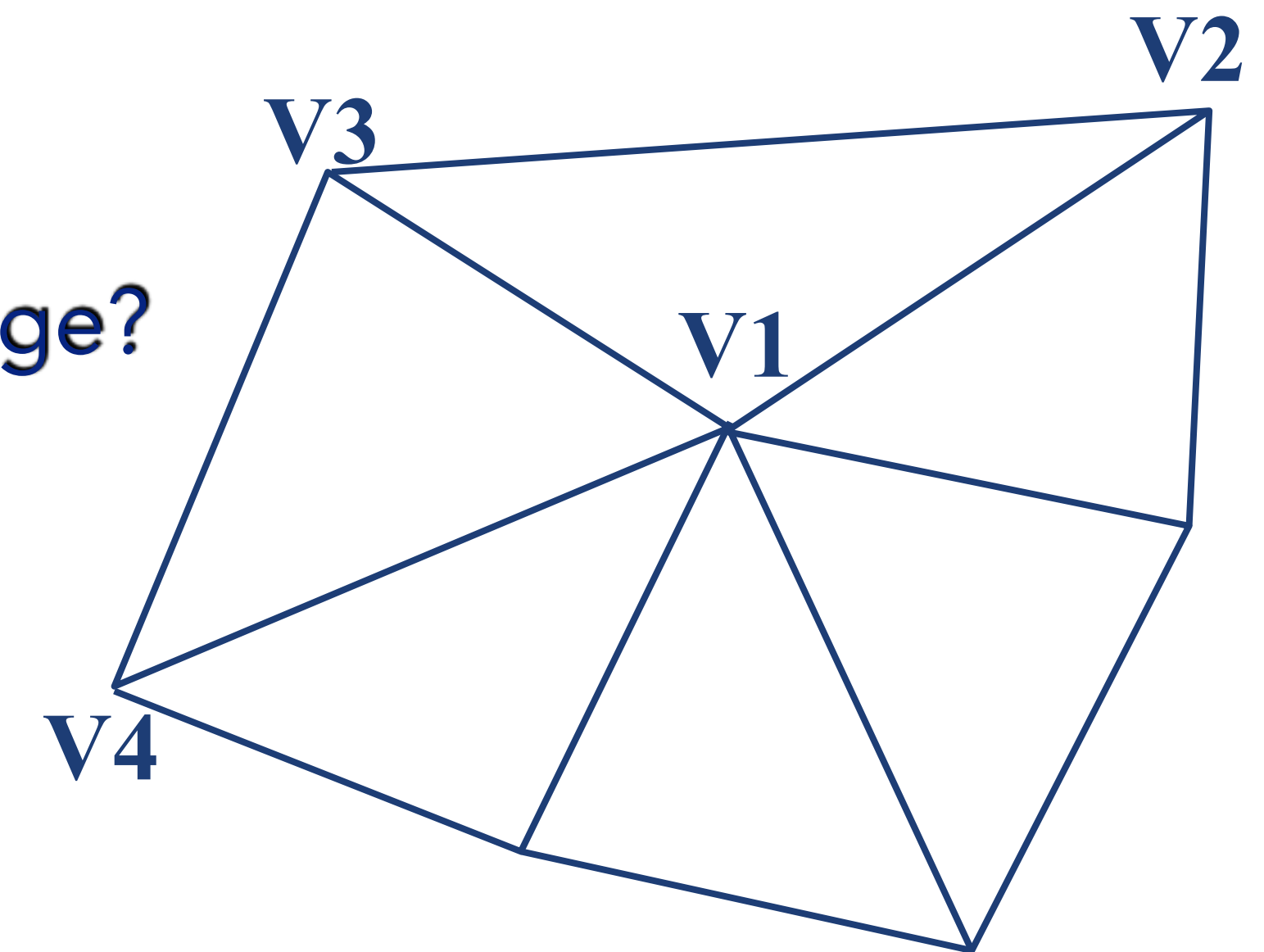


Vertex & Topology

- Indexed polygons

- ➔ Find neighbors?

- polygons sharing a given vertex? polygon sharing an edge?



Vertices

1. **$V1_x V1_y V1_z$ + additional attributes**
2. **$V2_x V2_y V2_z \dots$**
3. **$V3_x V3_y V3_z \dots$**
4. **$V4_x V4_y V4_z \dots$**
- ...

Polygons

1	2	3
1	2	4
...		

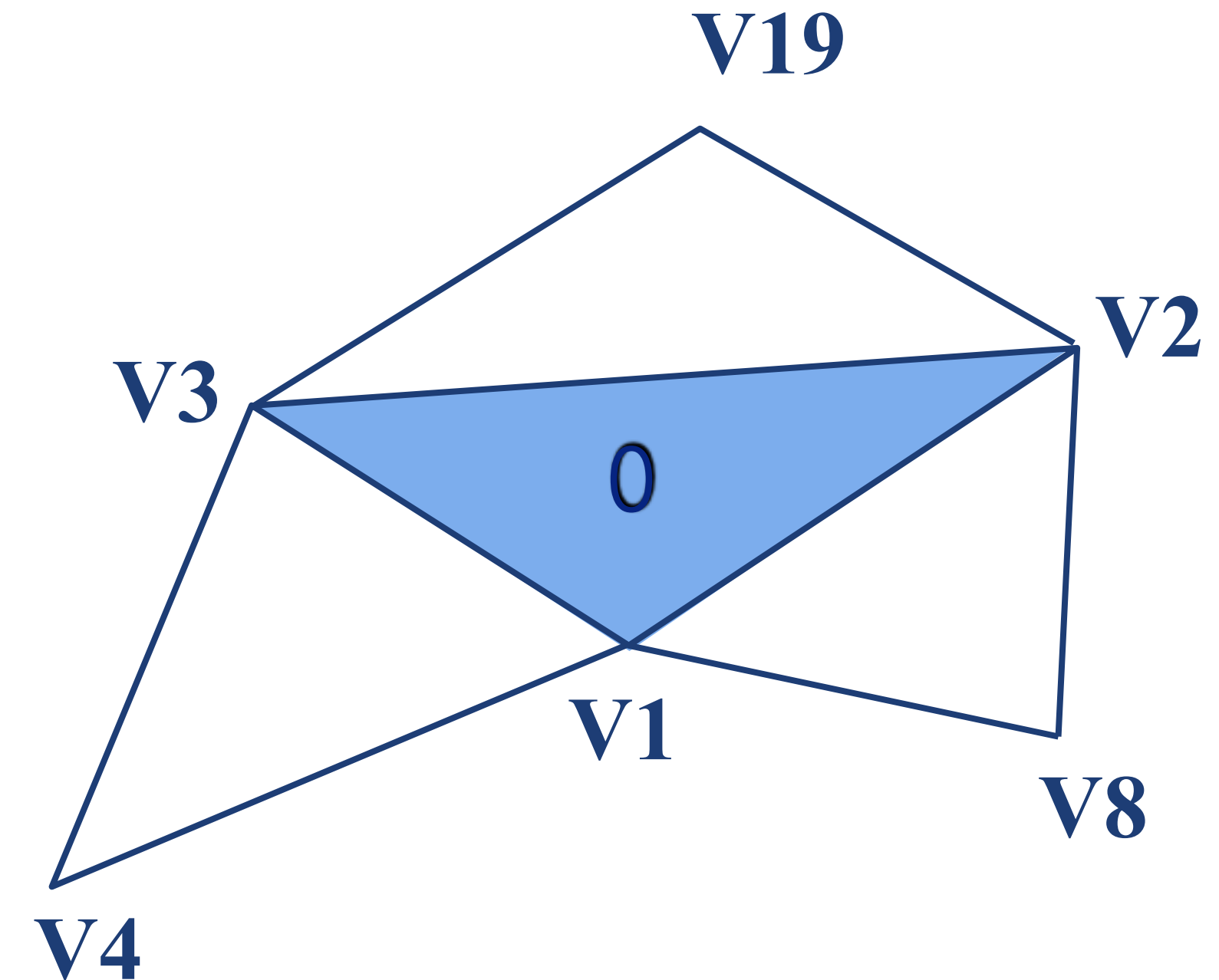


Triangle With Adjacency

- Index to neighbors

Vertices

1. **$V1_x V1_y V1_z$ + additional attributes**
2. **$V2_x V2_y V2_z \dots$**
3. **$V3_x V3_y V3_z \dots$**
4. **$V4_x V4_y V4_z \dots$**
- ...



Triangles with Adjacency

1 8 2 19 3 4
...

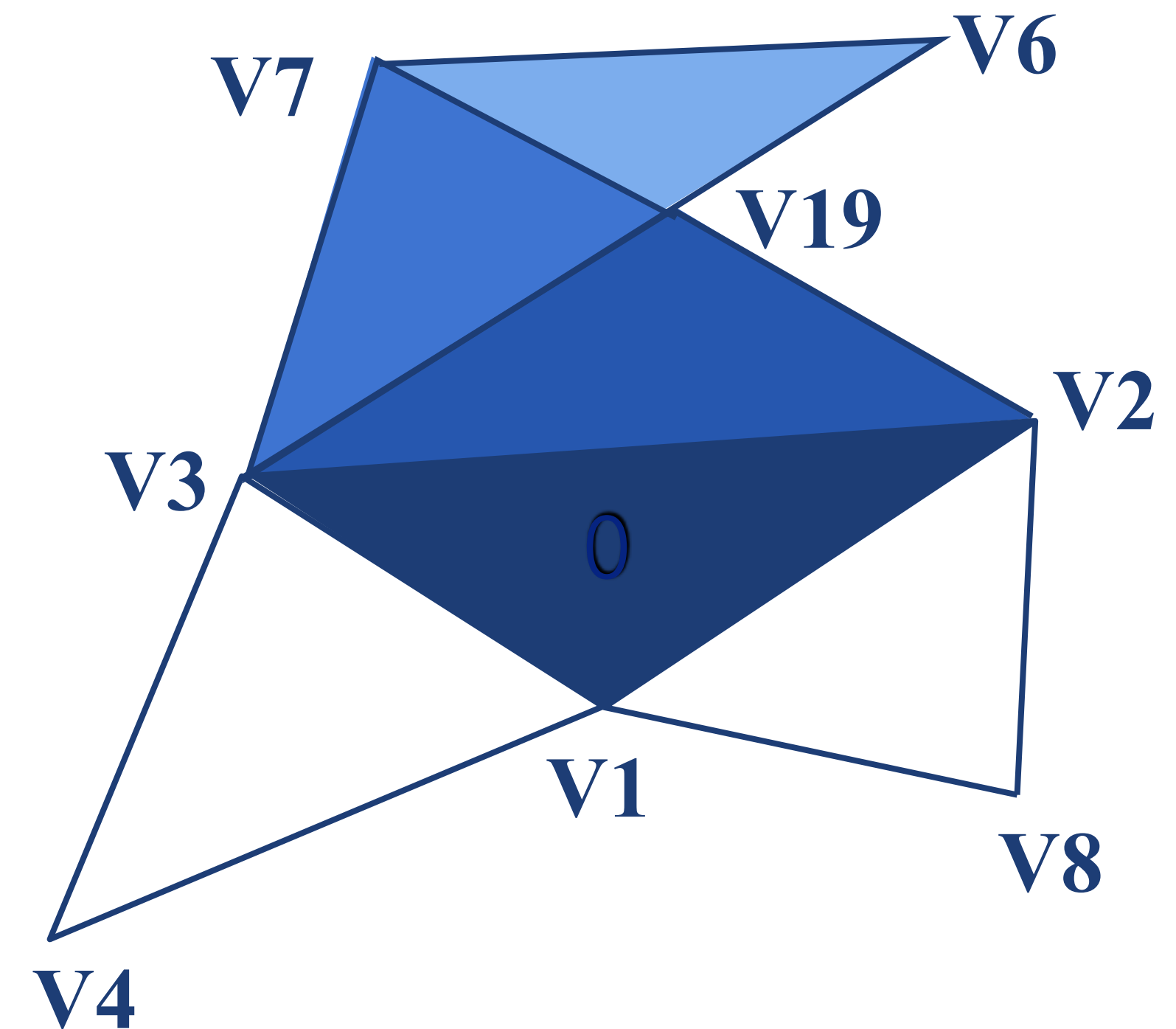


Triangle Strip

- Index to neighbors

Vertices

1. **$V1_x V1_y V1_z$ + additional attributes**
2. **$V2_x V2_y V2_z \dots$**
3. **$V3_x V3_y V3_z \dots$**
4. **$V4_x V4_y V4_z \dots$**
- ...



Triangle Strip

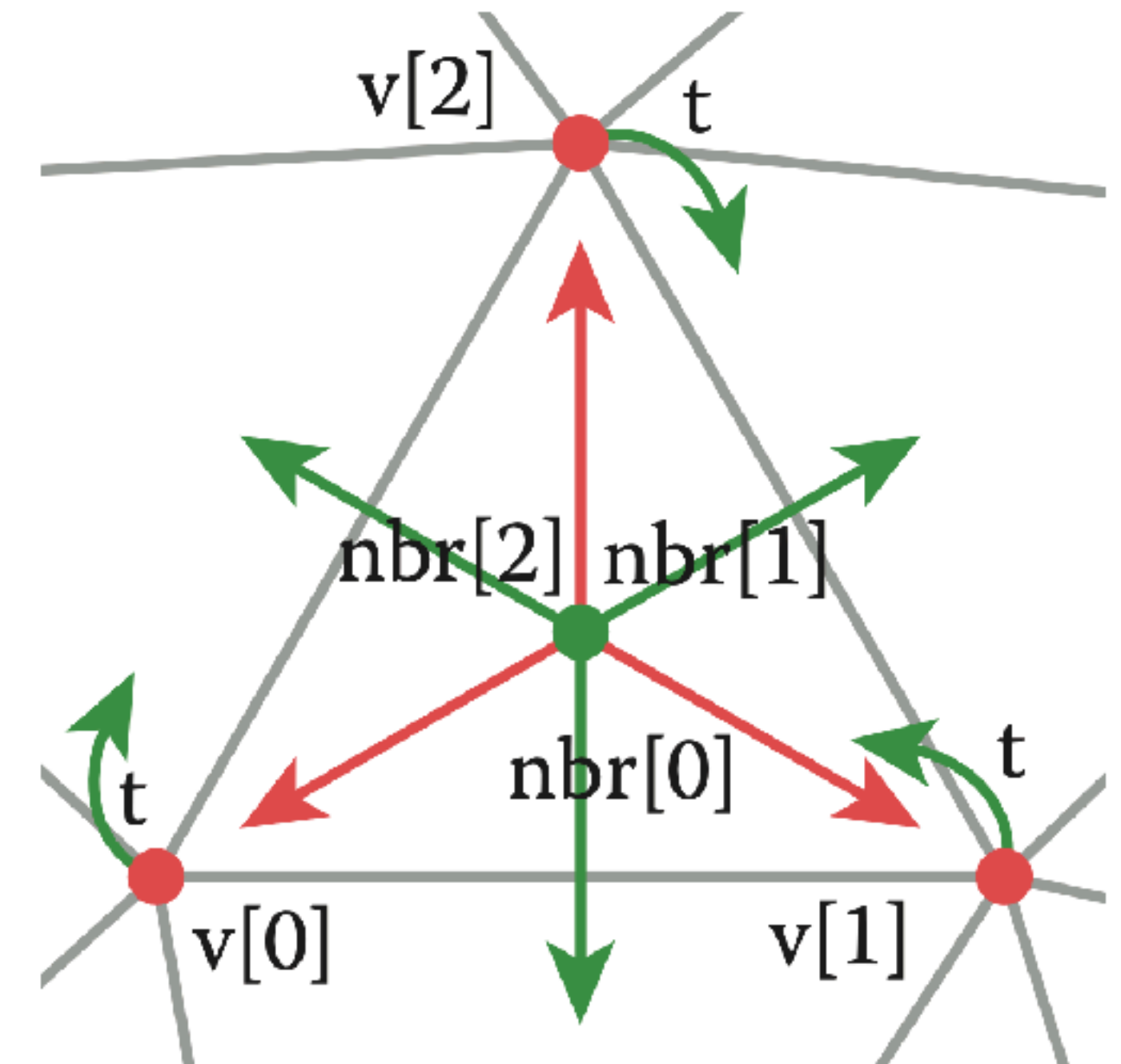
1 2 3 19 7 6
...

Triangle neighbour data structure



```
Vertex {  
    Point position;  
    Triangle *triangle;  
}  
  
Triangle {  
    Vertex *vertices[3];  
    Triangle *neighbors[3];  
}
```

Instead of pointers, just store index
into the vertex/triangle array.



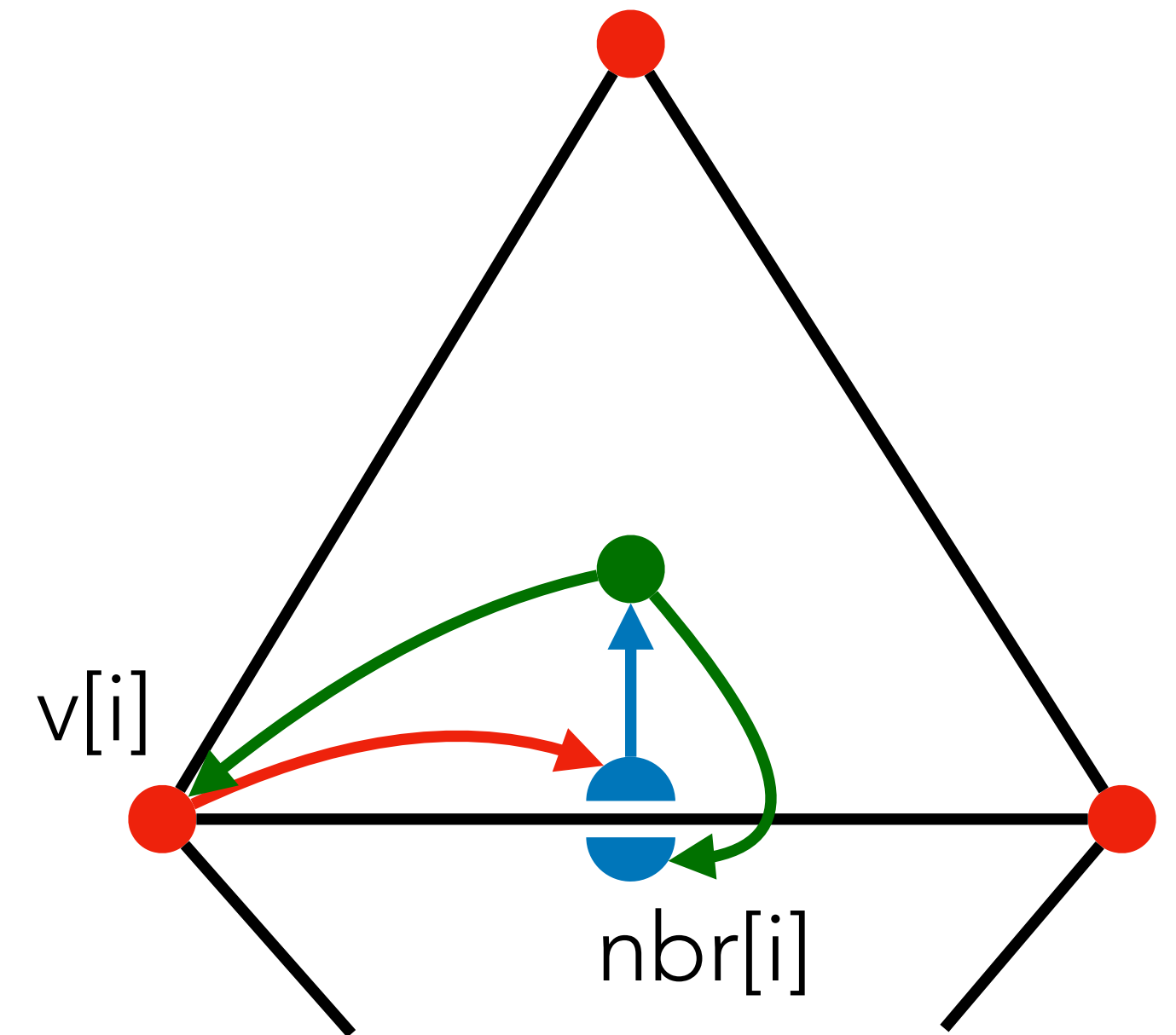
Edge-based data structure



```
Vertex {  
    Point position;  
    Edge *edge;  
}
```

```
Edge {  
    Triangle *triangle;  
    int index;  
}
```

```
Triangle {  
    Vertex *vertices[3];  
    Edge *neighbors[3];  
}
```





Half-edge data structure

```
HalfEdge {  
    HalfEdge *pair, *next;  
    Vertex *head;  
    Face *left;  
}  
  
Vertex {  
    HalfEdge *halfEdge;  
}  
  
Face {  
    HalfEdge *halfEdge;  
}
```

