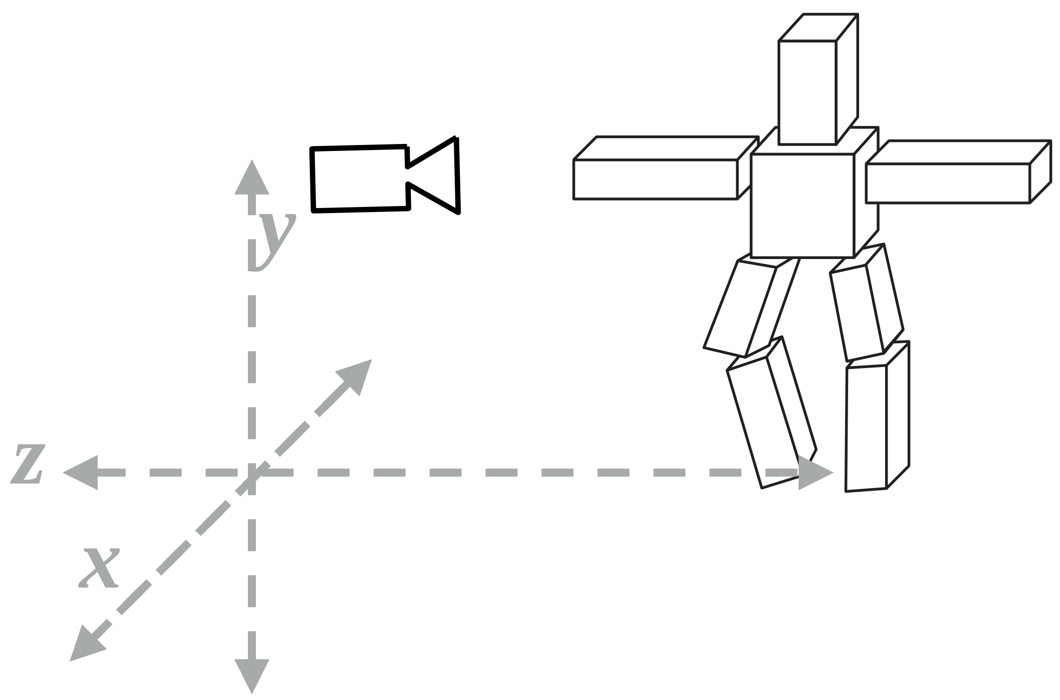
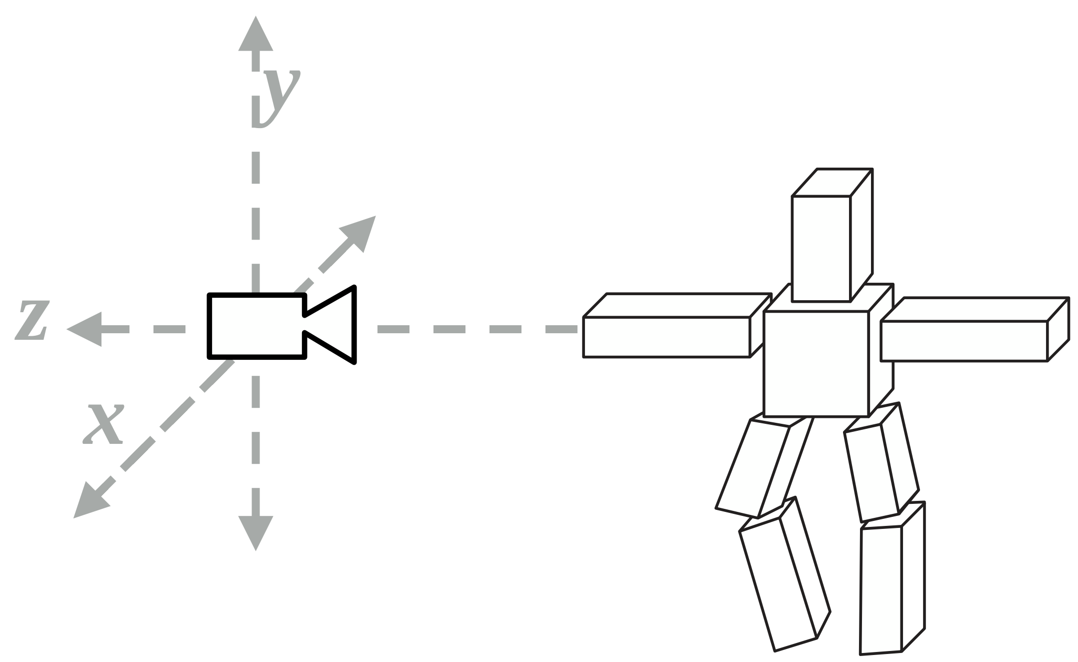
The Rasterization Pipeline

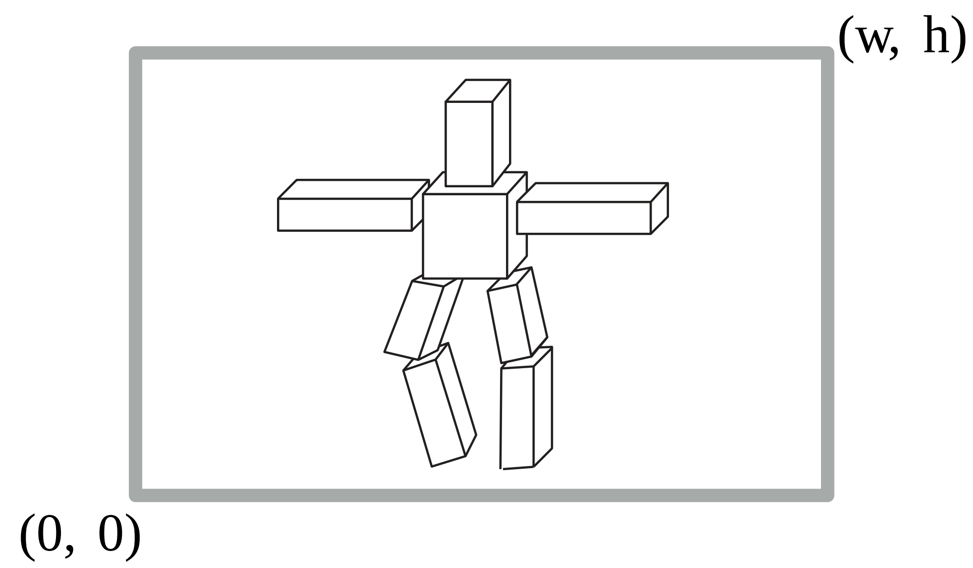
What We’ve Covered So Far



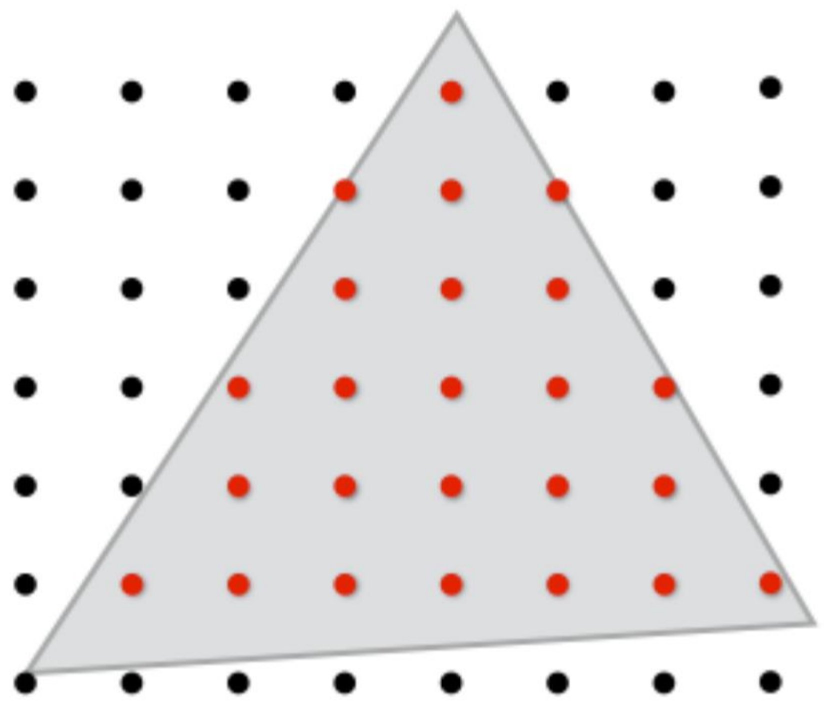
Position objects and the camera in the world



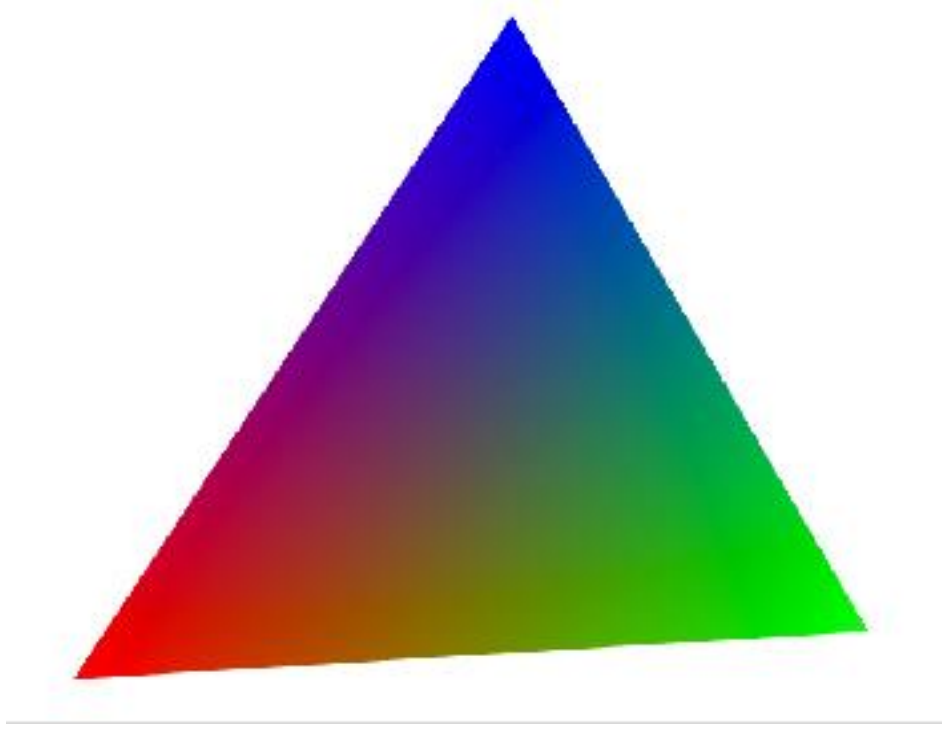
Compute position of objects relative to the camera



Project objects onto the screen



Sample triangle coverage



Interpolate triangle attributes

Sample texture maps

What Else Are We Missing?

Surface representations

- Objects in the real world exhibit highly complex geometric details

Lighting and materials

- Appearance is a result of how light sources reflect off complex materials

Camera models

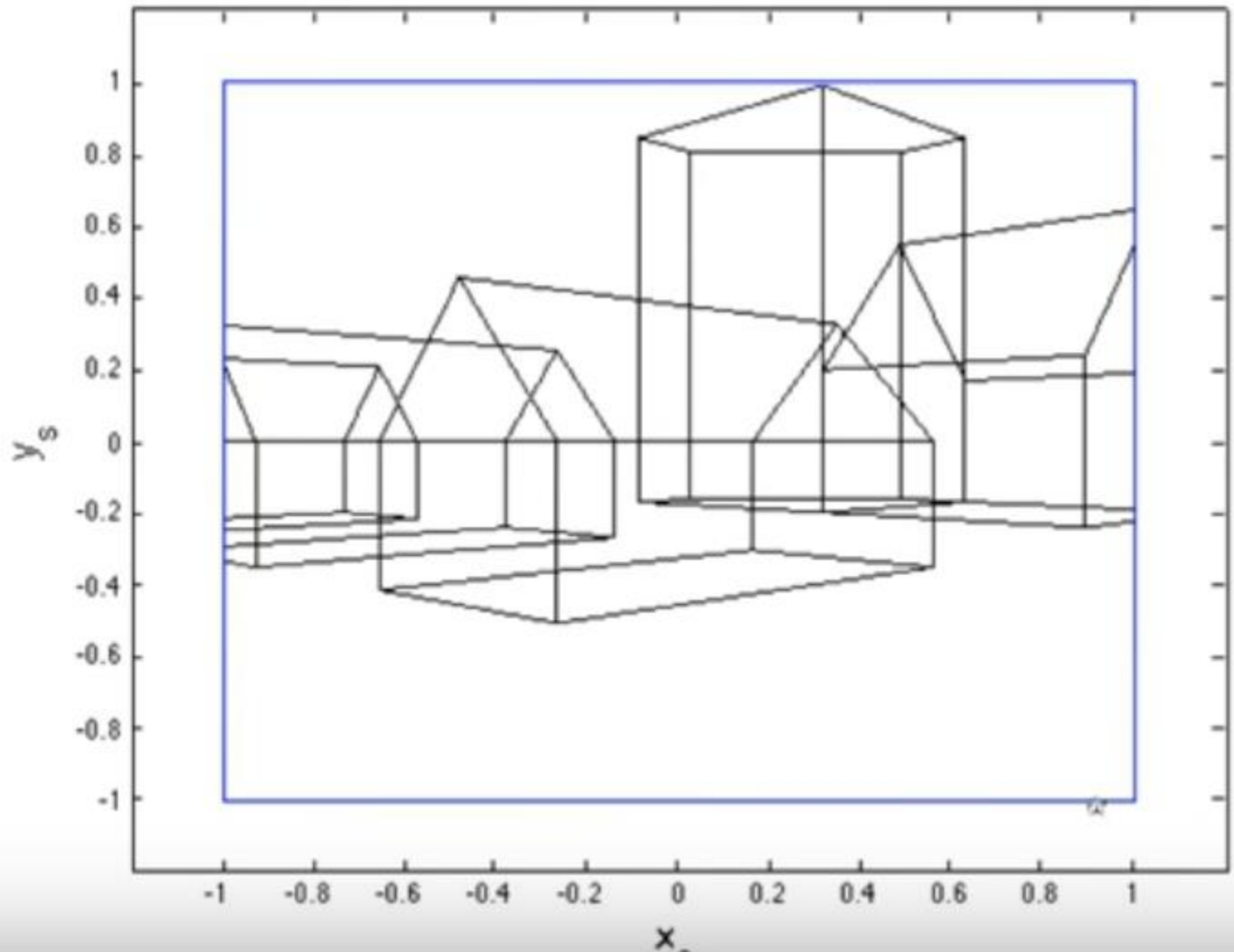
- Real lenses create images with focusing and other optical effects

Course Roadmap

Rasterization Pipeline

- Intro
- Rasterization
- Transforms & Projection
- Texture Mapping
- Today: Visibility, Shading, Overall Pipeline

Visibility

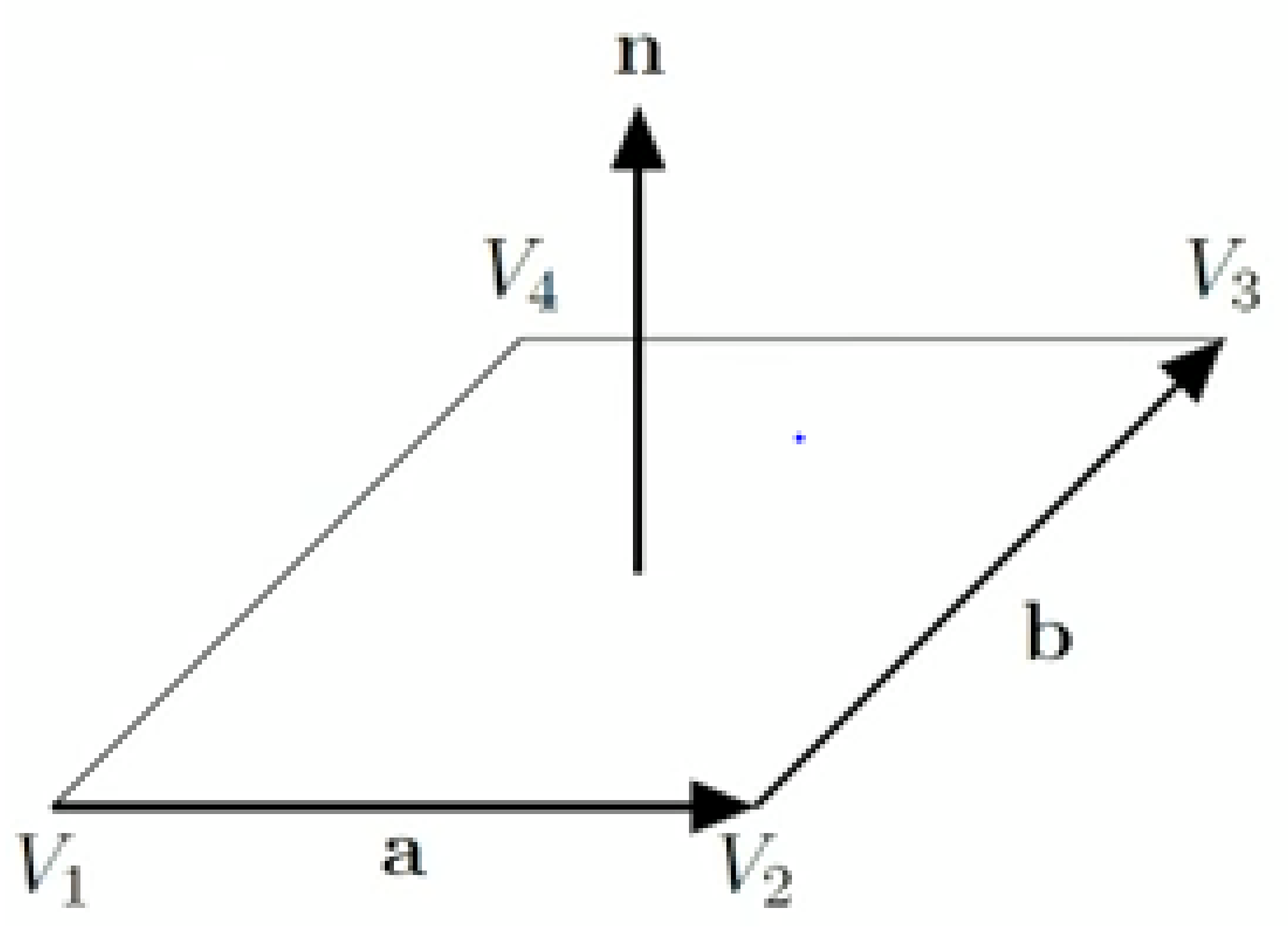


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
y_s
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
x_s

Classification of Visible Surface Detection

- **Object space method:**
  - –Compares object and parts of object to each other within the scene definition to determine which surface as whole is visible.
- **Image Space method:**
  - –Visibility is decided point by point at each pixel position on the projection plane.

Computing Surface Normal



The normal vector points perpendicular away from the plane or polygon.

We use cross product between any vector.

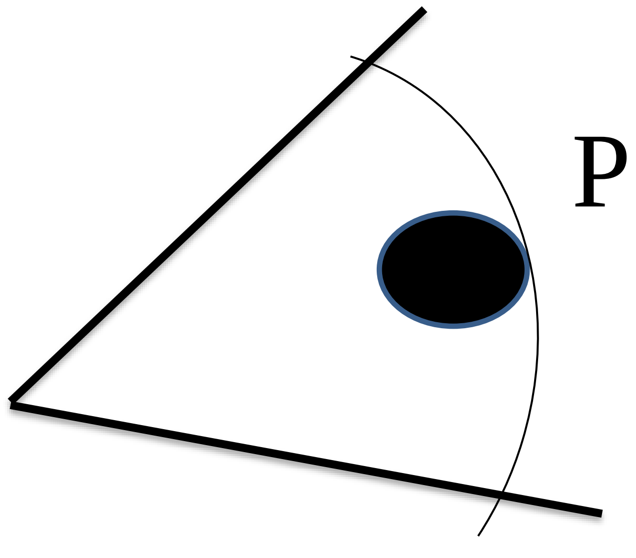
n=a x b

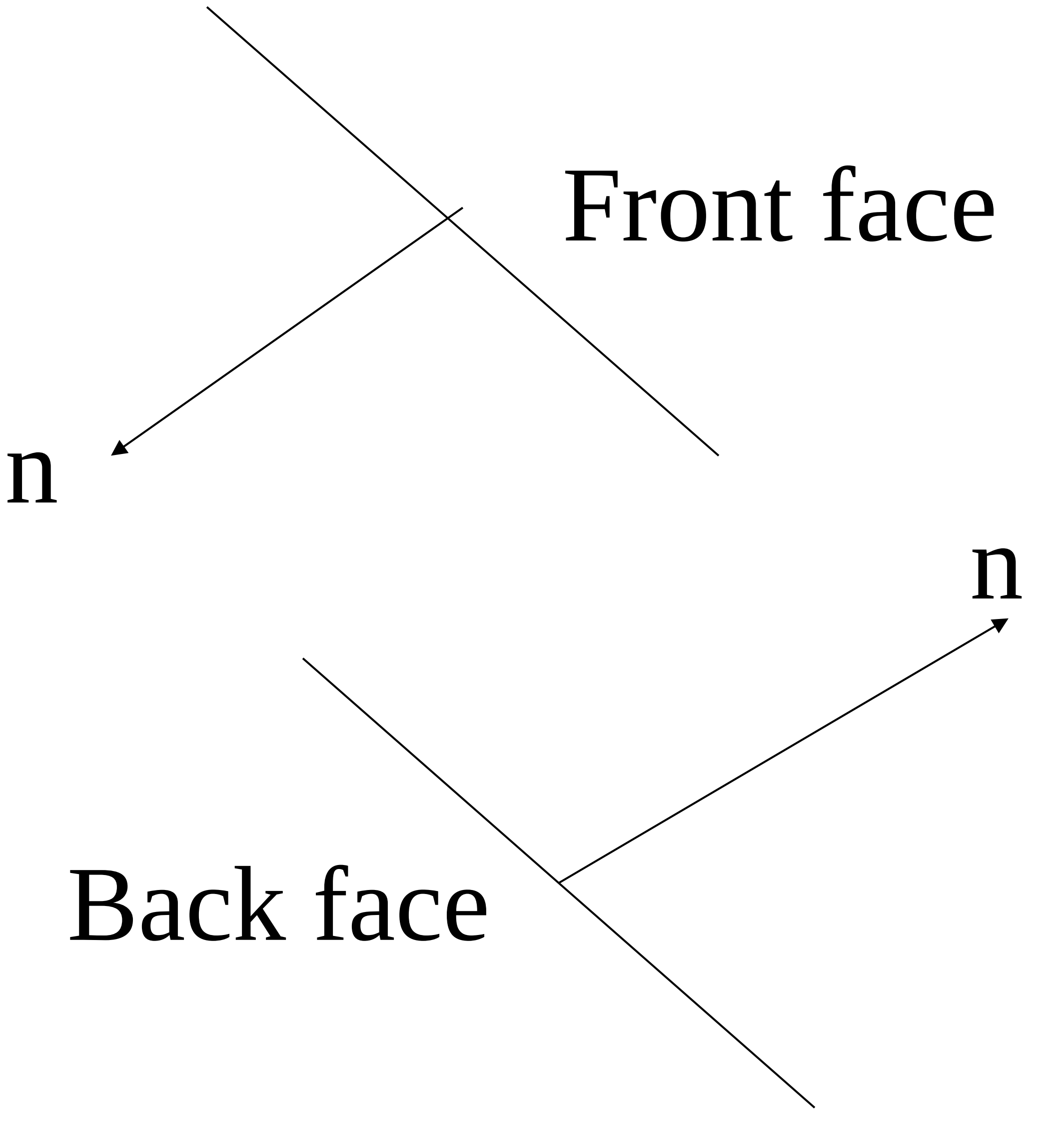
$$\mathbf{n} = (\mathbf{V_2} - \mathbf{V_1}) \times (\mathbf{V_3} - \mathbf{V_2})$$

n=a x b

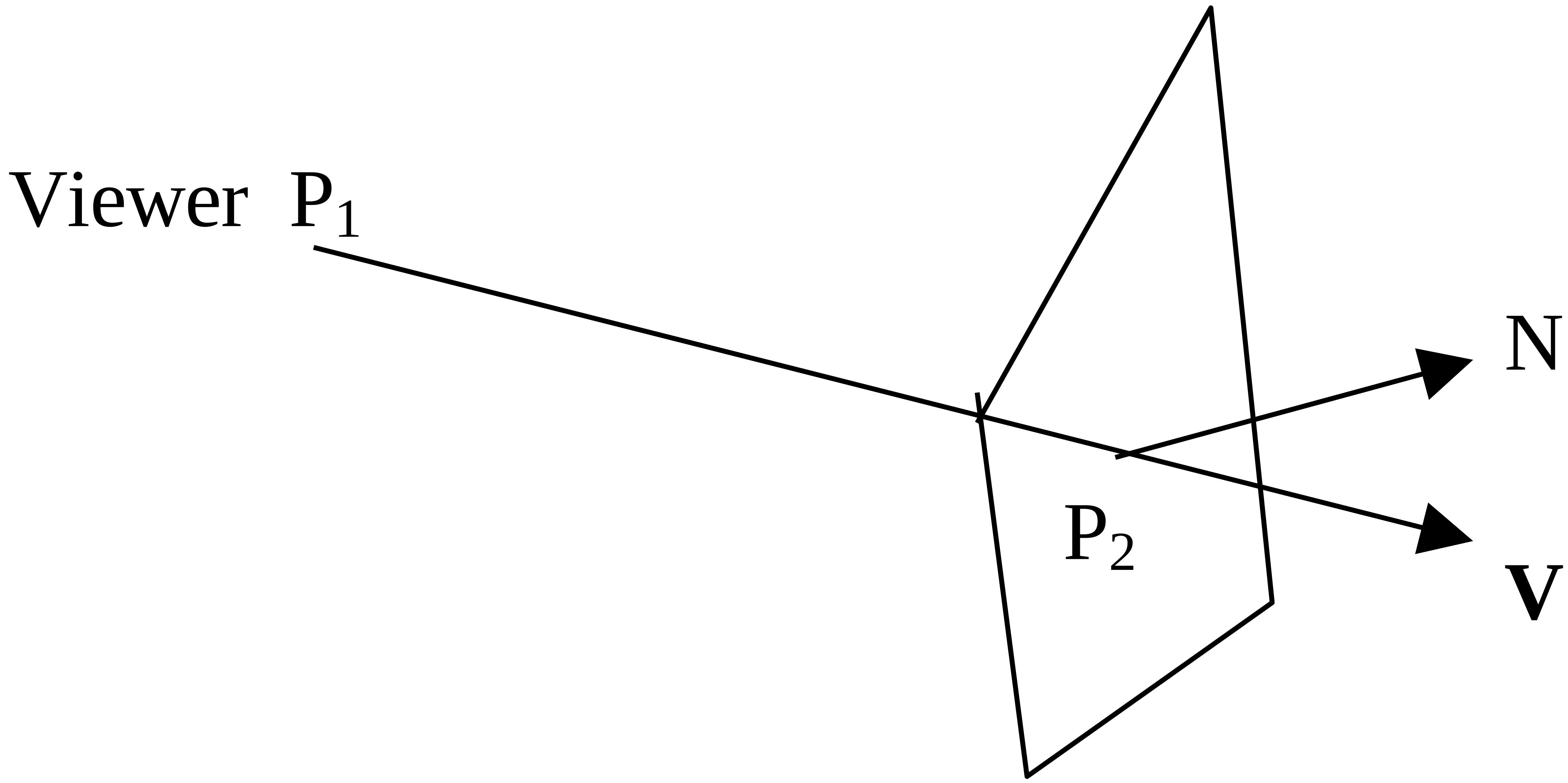
Back Face Detection

- A polygon is front facing if its surface normal vector to pointing towards the viewpoint, else it is back facing.





Back Face Detection

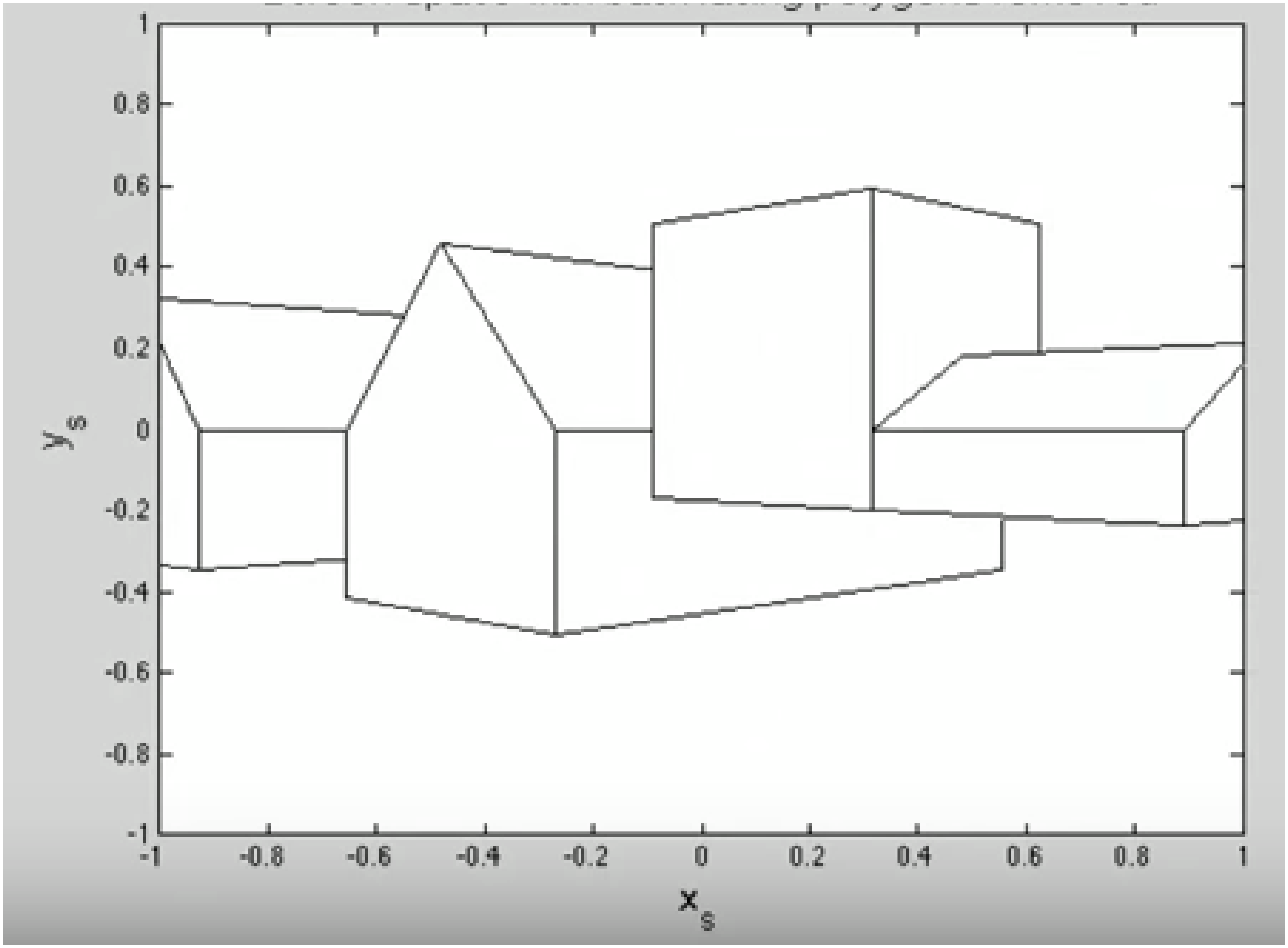


V.N=|V||N| cosΘ

V.**N** > 0 ⇒ The polygon is hidden

V.N<0 ⇒ The polygon is visible.

Back Face Detection



Back Face Detection- Limitations

- Requires specific ordering of the vertices in the polygon table to determine the outward normal direction.
- The algorithm will work only with **convex** objects.
- A polygon is either completely displayed, or totally eliminated from the display.

Back Face Detection Algorithm

Algorithm 4 Back face culling

Require: Vertex co-ordinates of polygons and an viewpoint P
  for all polygons in the virtual world **do**
    calculate the normal vector $\mathbf{n}$ of the current polygon
    calculate the centre C of the current polygon
    calculate the viewing vector $\mathbf{v} = C - P$
    if $\mathbf{v} \cdot \mathbf{n} < 0$ **then**
      render current polygon
    end if
  end for

Back Face Detection OpenGL

Defining a *front face* as the face with CCW ordering of vertices (Default):

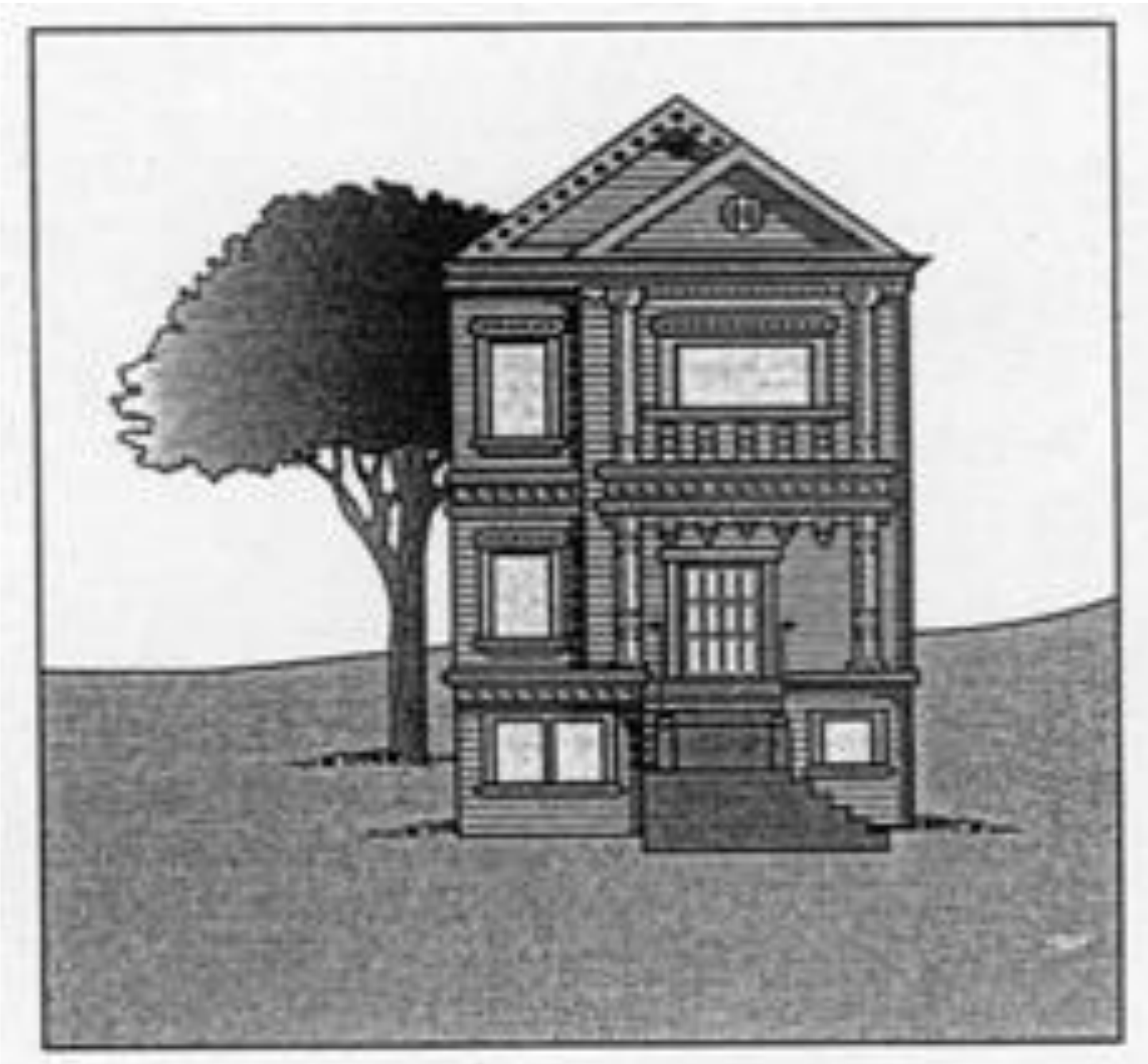
```
glFrontFace(GL_CCW);
```

Enabling polygon culling:

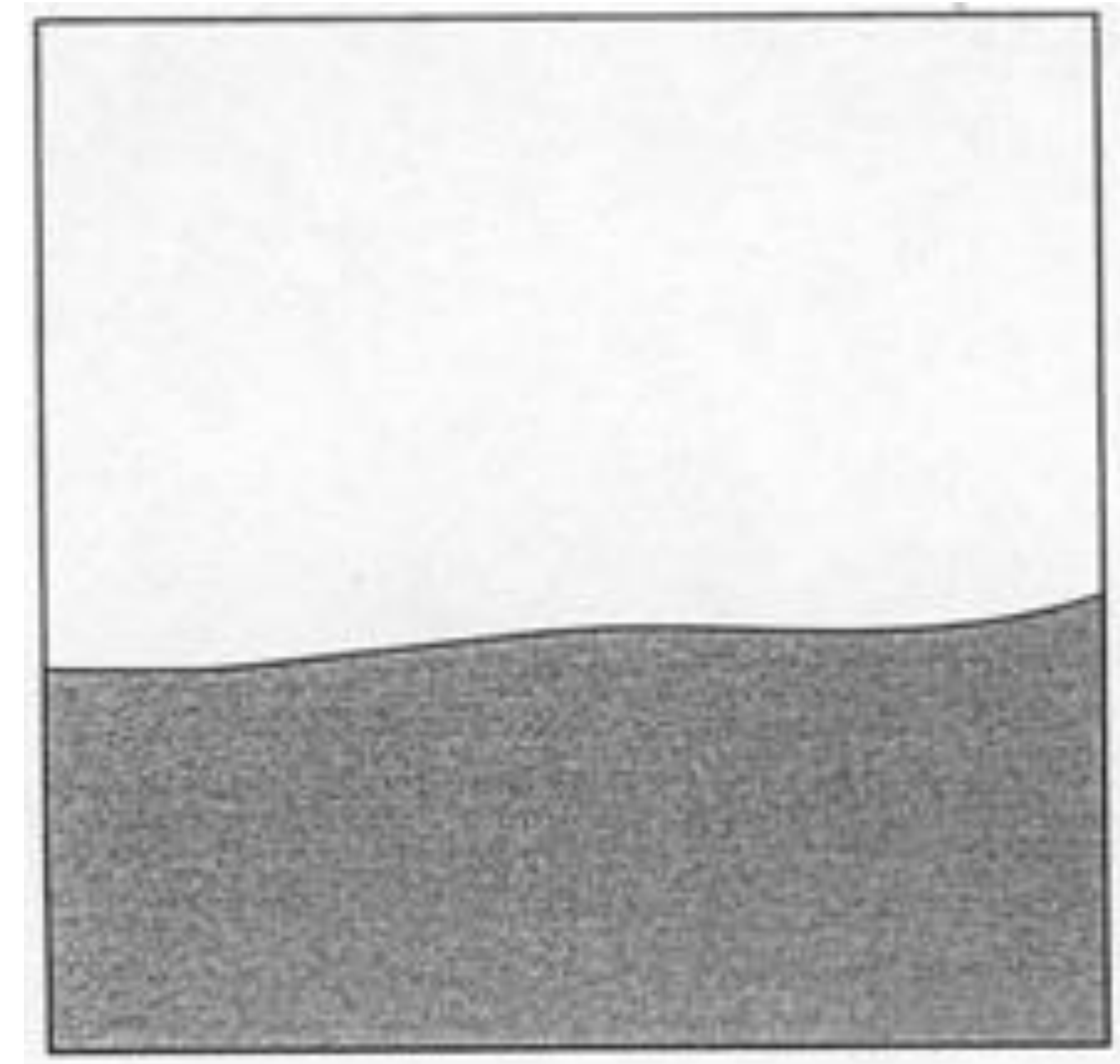
```
glEnable(GL_CULL_FACE);
```

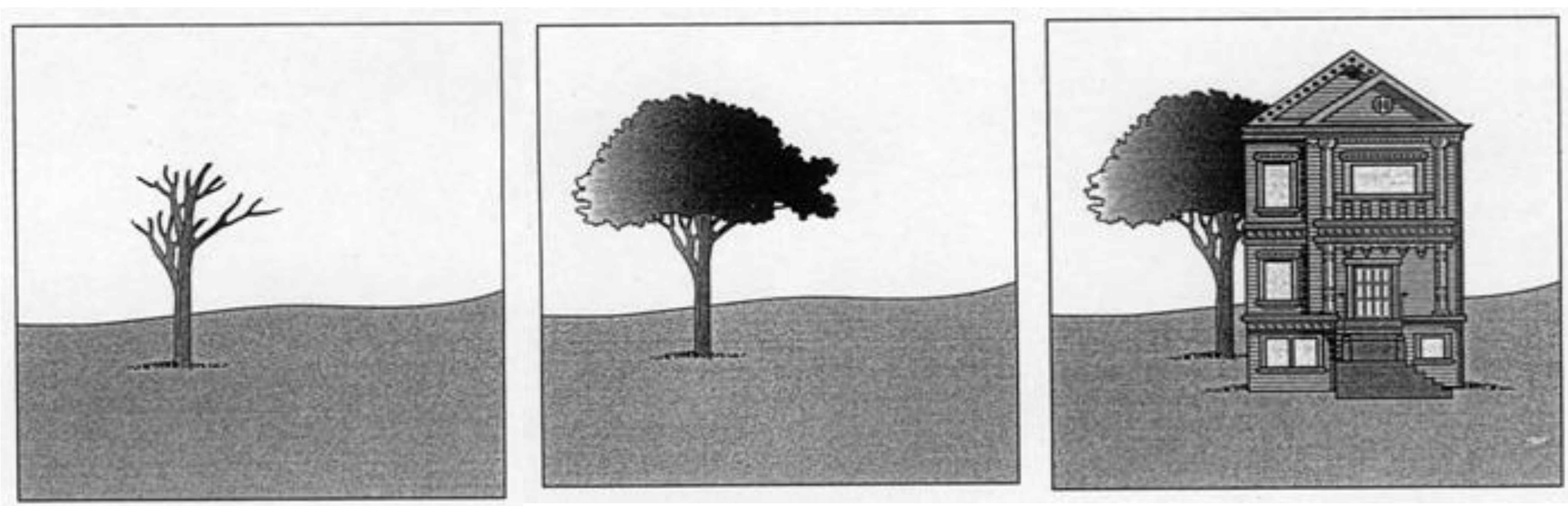
Discard back facing polygons:

```
glCullFace(GL_BACK);
```



By Painter’s Algorithm

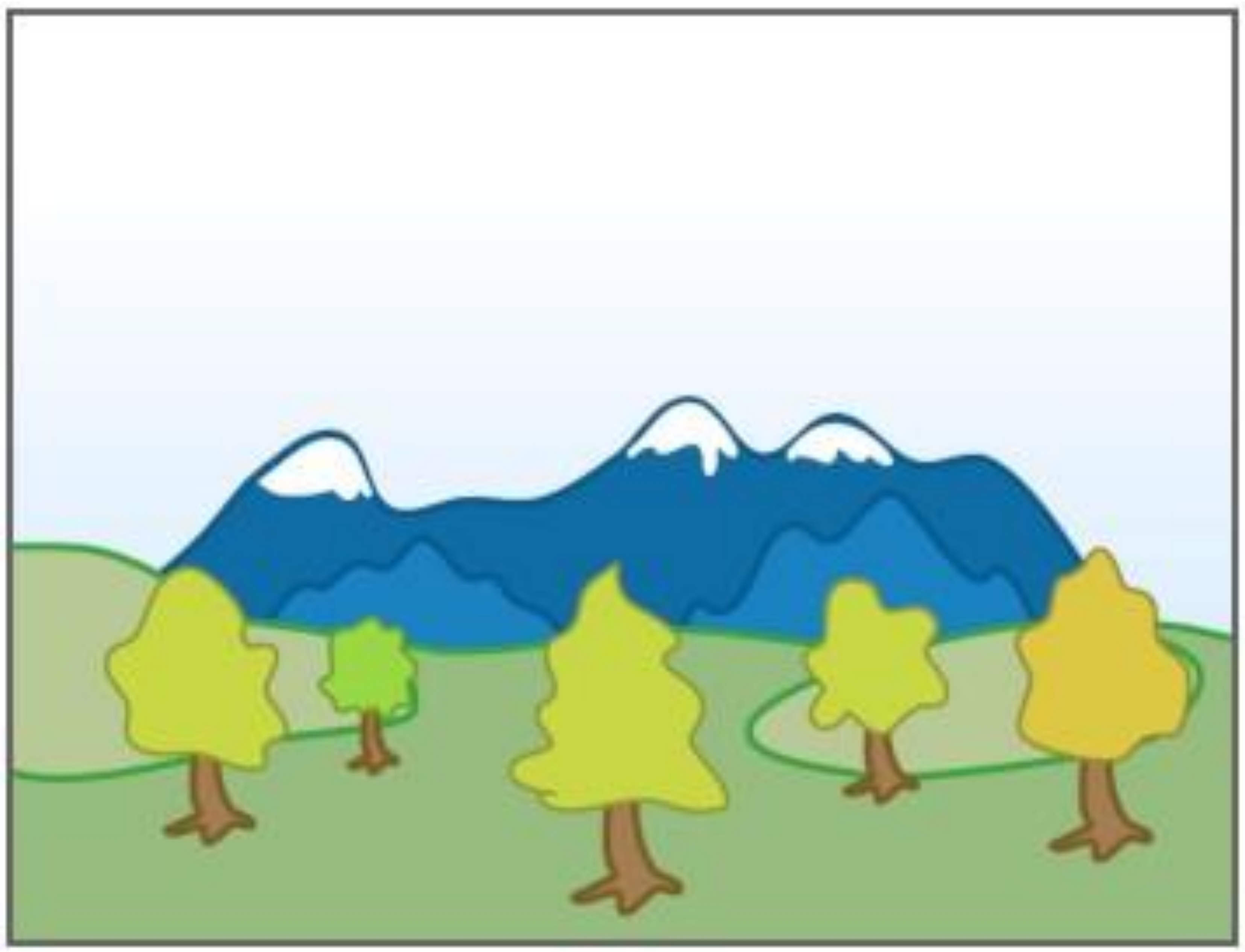




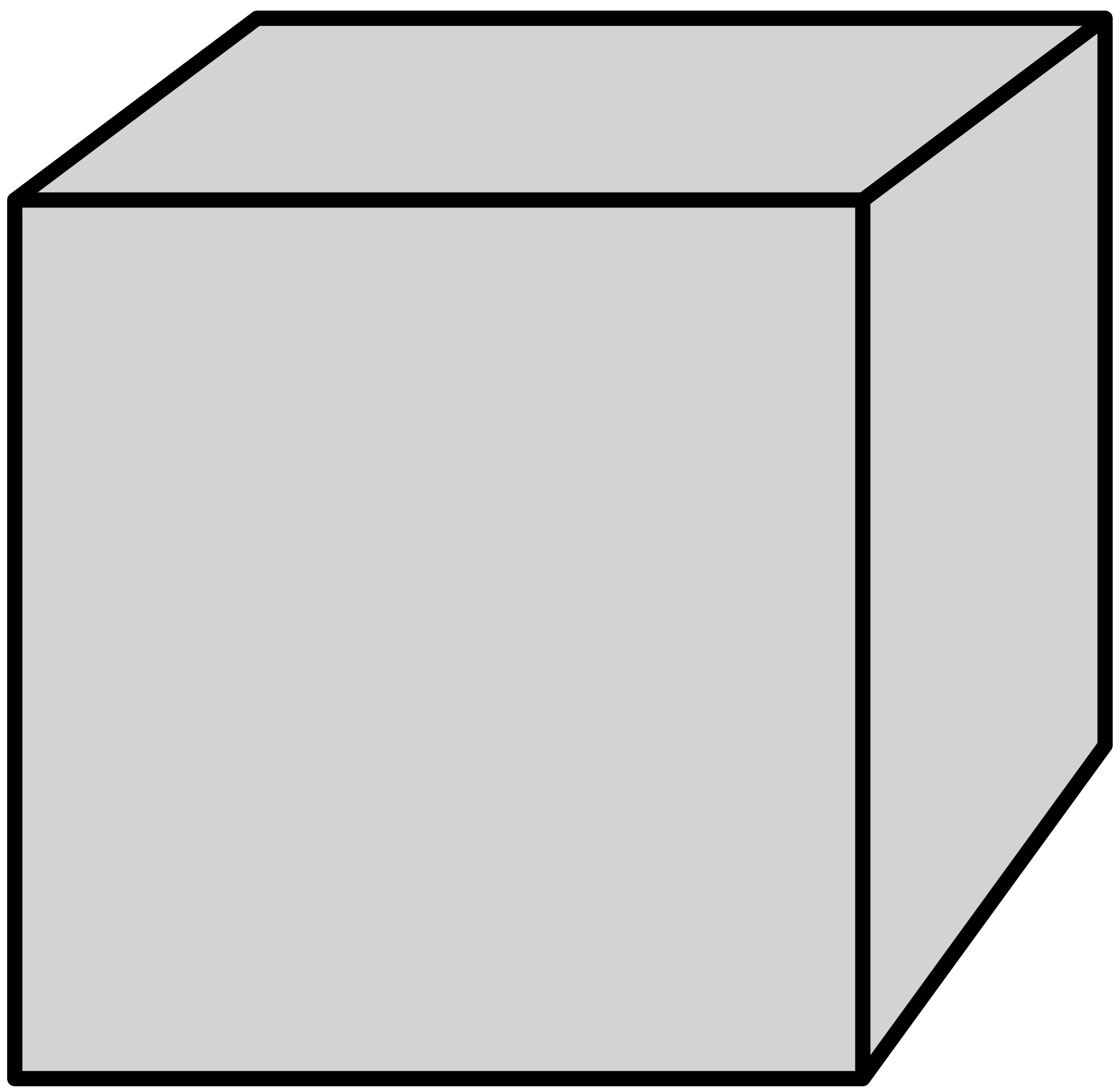
Painter's Algorithm

Inspired by how painters paint

Paint from back to front, overwrite in the framebuffer



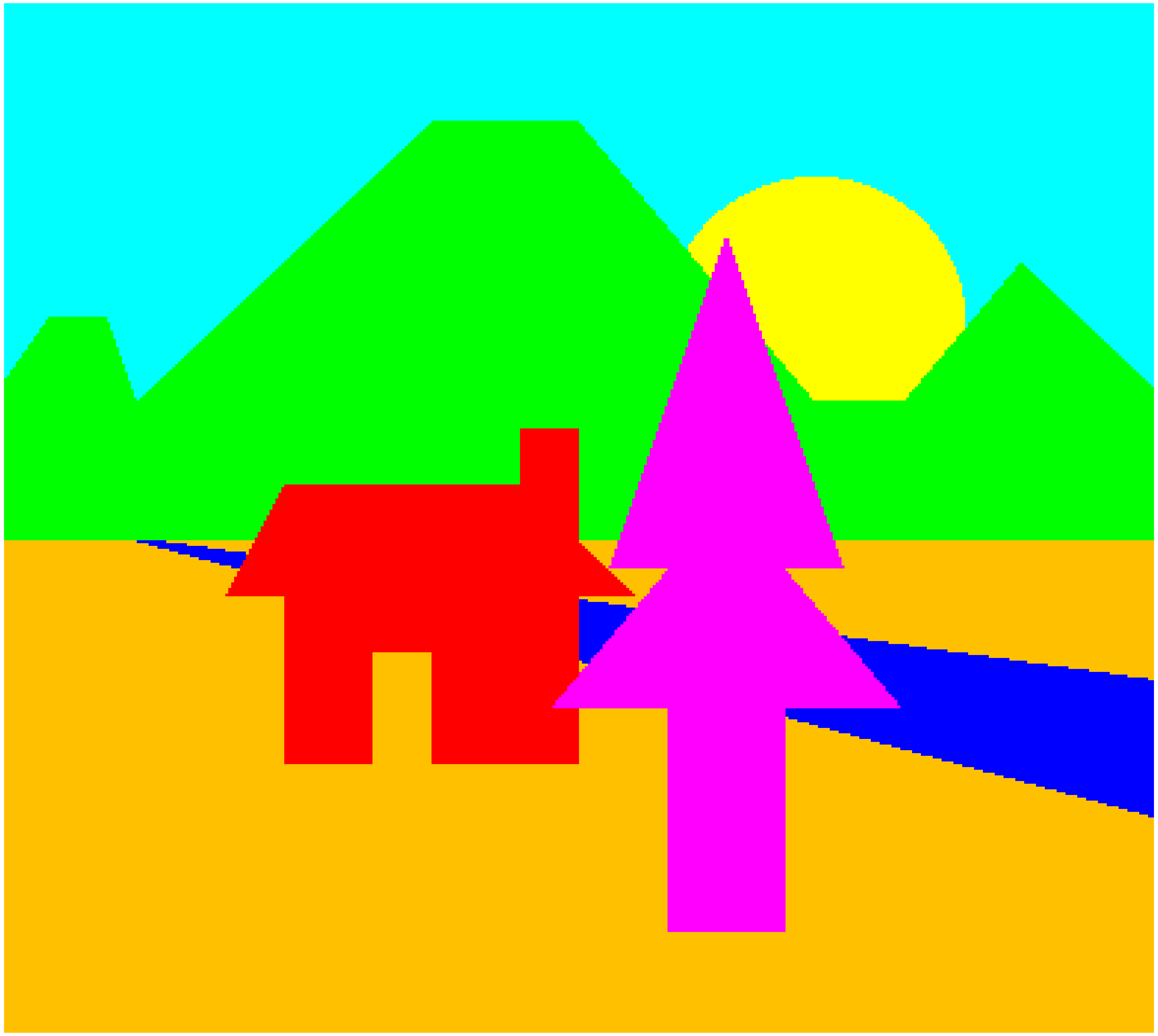
[Wikipedia]



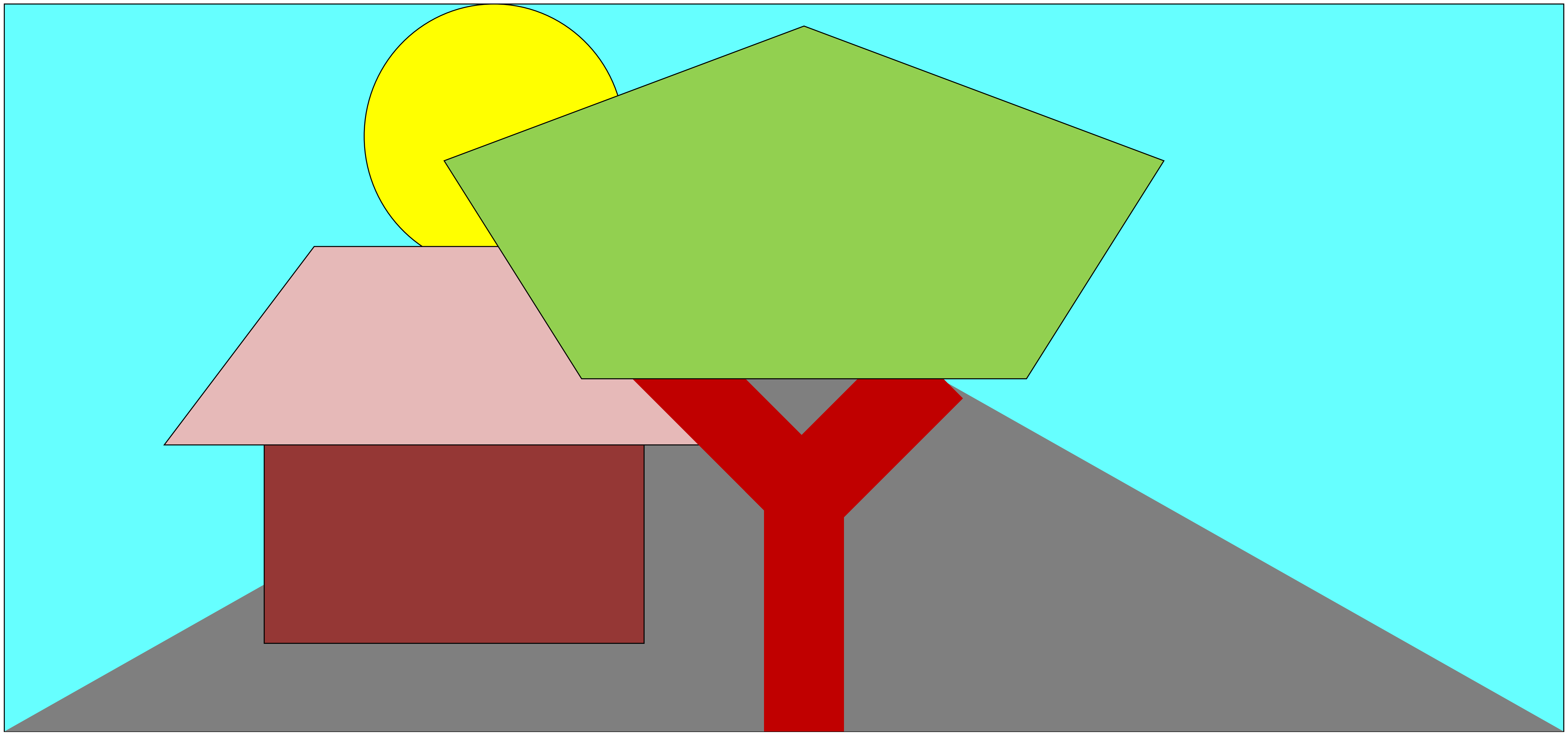
Painter's Algorithm

- **Basic Steps:**
  - **Surface are sorted in order of decreasing depth.**
  - **Fill polygons in the sorted order.**

Painter's Algorithm



Painter's Algorithm

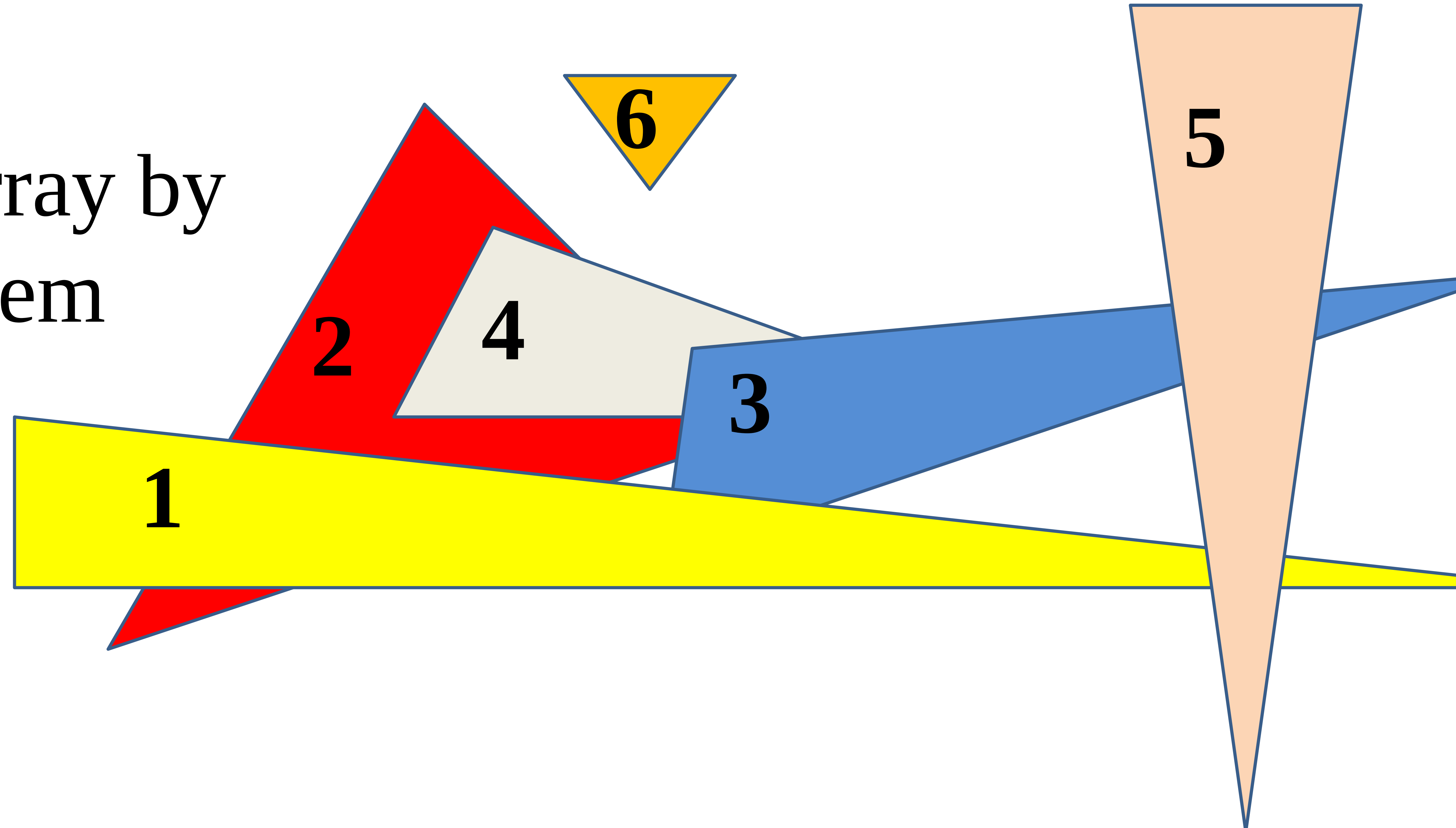


Painter's Algorithm

STEP 1

Store the polygon in array by uniquely numbering them

1	2	3	4	5	6



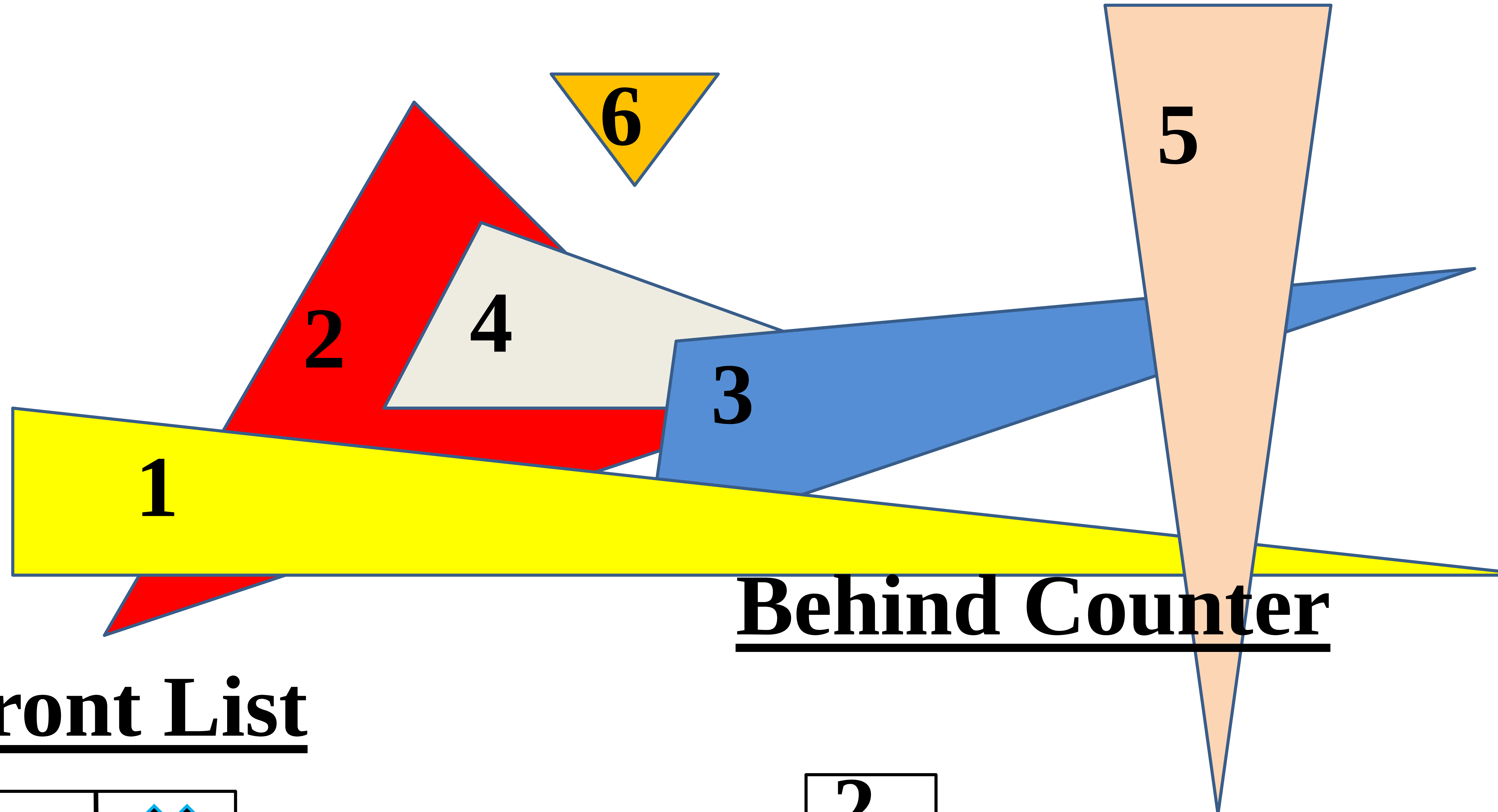
STEP 2

For each polygon maintain a **linked list** named as **front list** which will contain the polygon no. in front of it in a sequence

STEP 3

For each polygon maintain a **counter** named as **behind counter** which will contain the no. of polygon behind to it

Initialization



Polygon Array	Front List	Behind Counter
1	5 ✖	2
2	4 3 1 ✖	0
3	1 5 ✖	2
4	3 ✖	1
5	✖	2
6	✖	0

STEP 4 (loop)

Repeat step 5 & step 6 till all polygon **behind counter is -1**

STEP 5

Draw all polygon's whose behind counter is 0

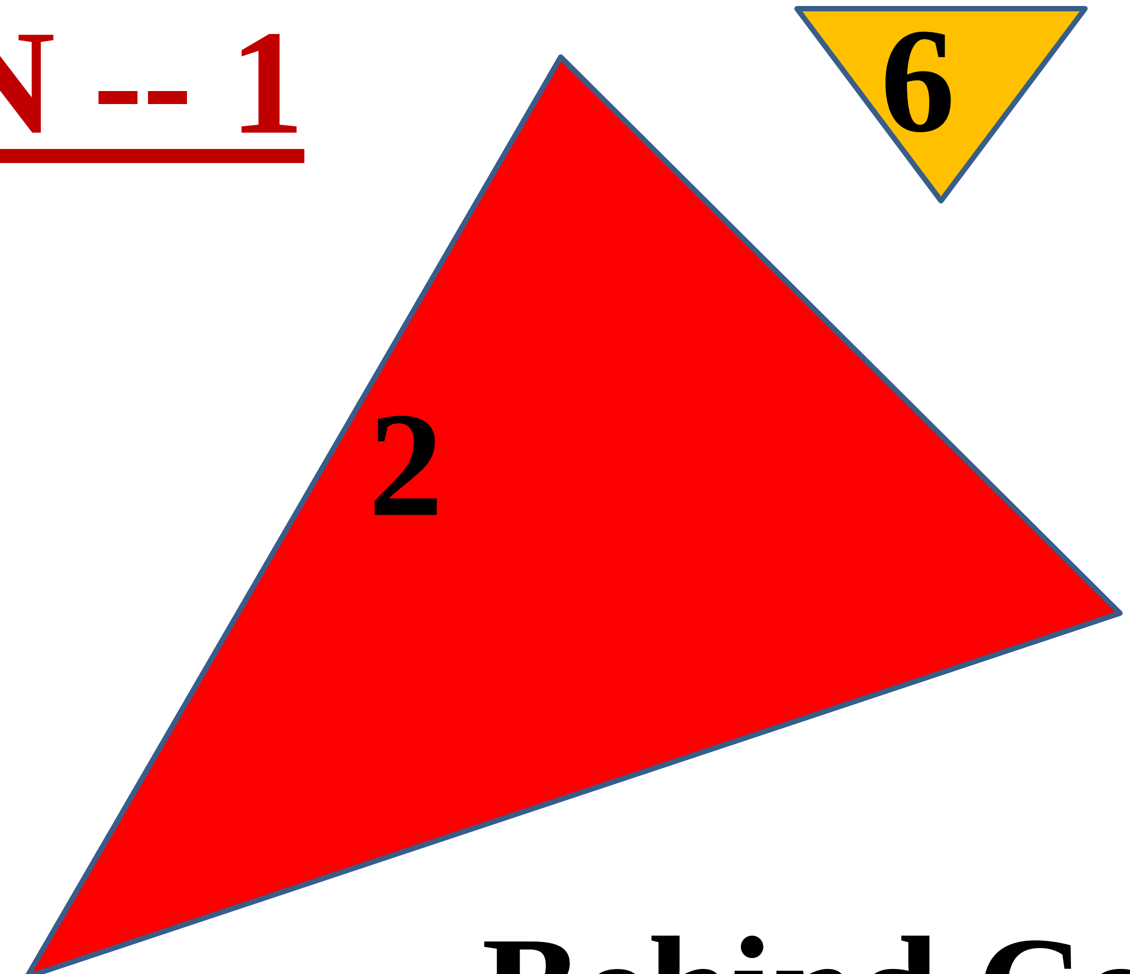
STEP 6

After drawing

a) Step through the polygon in the front list, decrease their behind counter by 1.

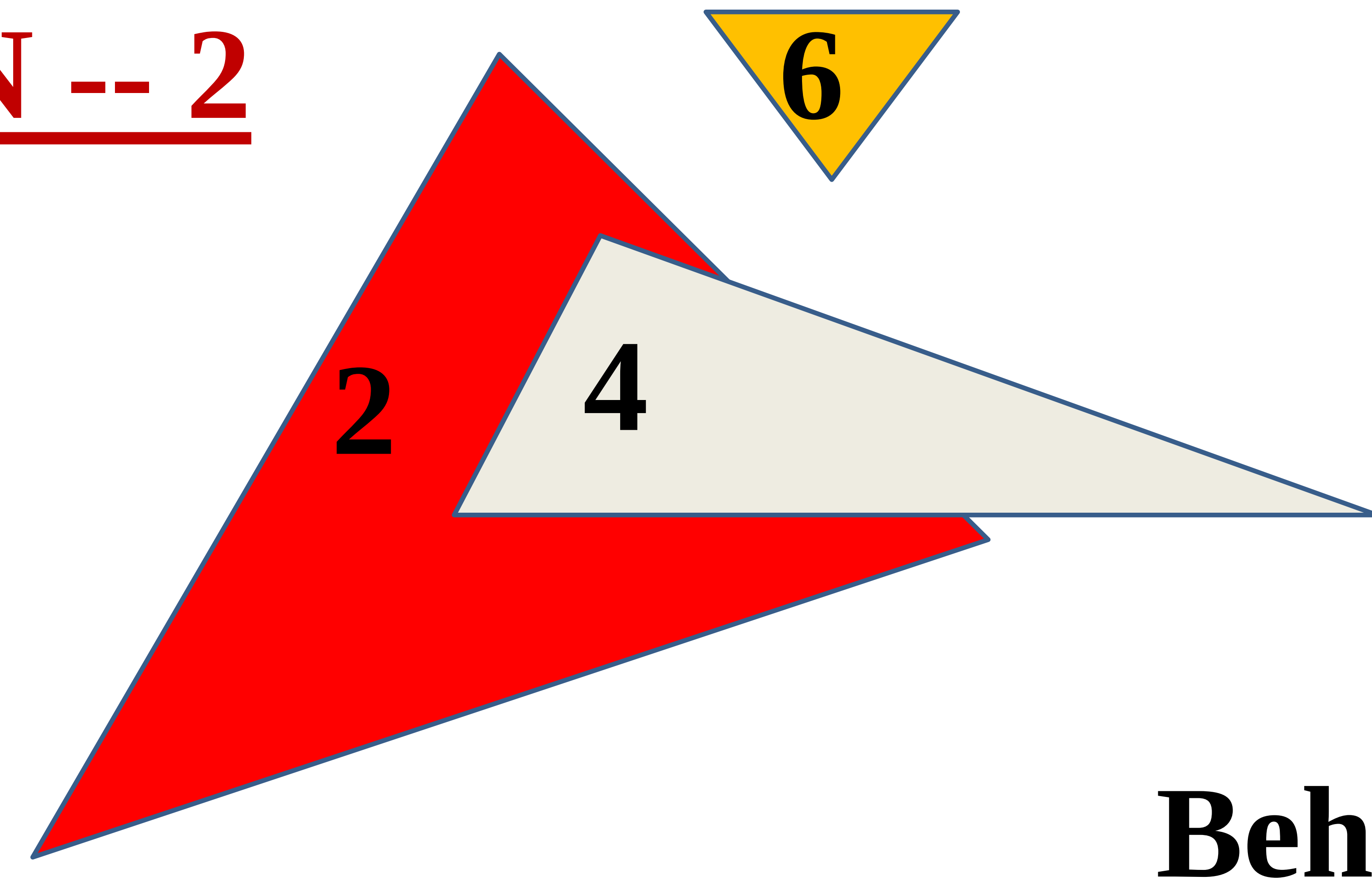
b) For drawn polygon make it's behind counter as **-1**

ITERATION -- 1



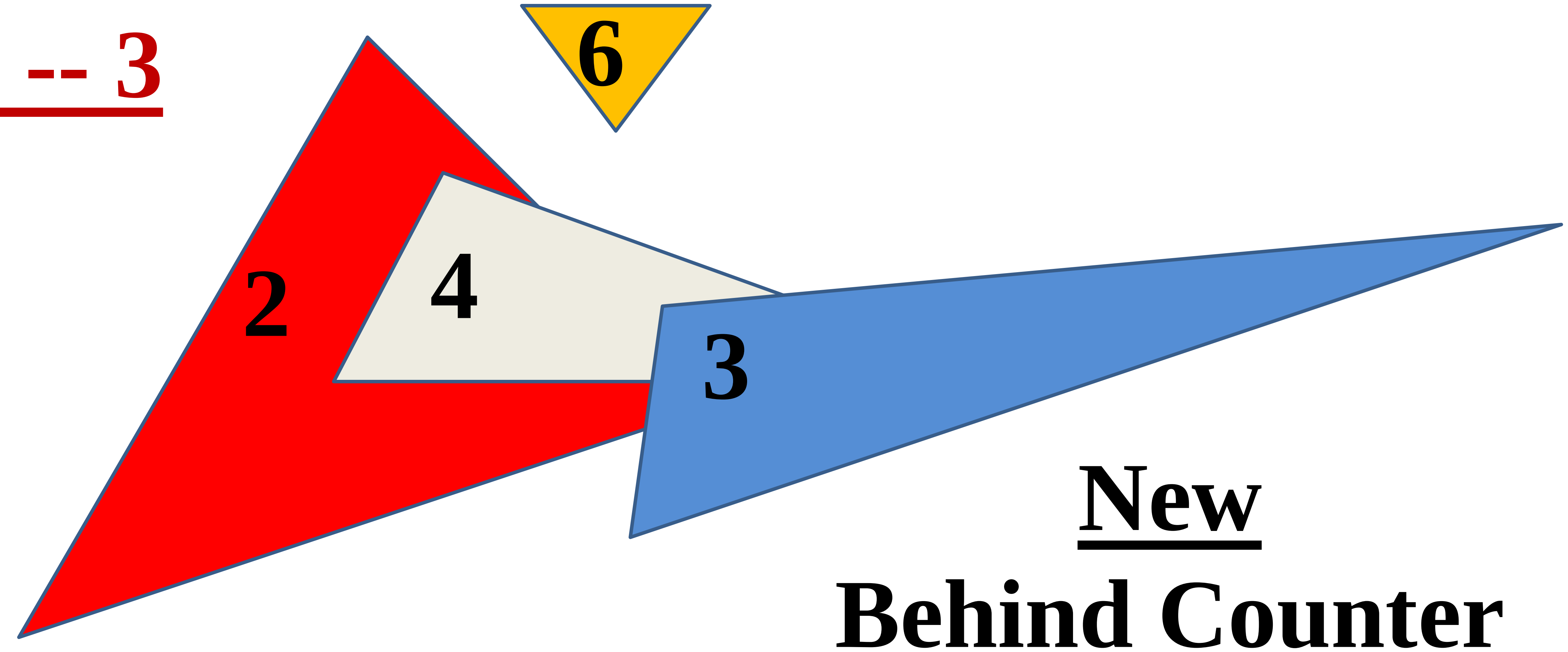
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5	2	1
2	4, 3, 1	0	-1
3	1, 5	2	1
4	3	1	0
5		2	2
6		0	-1

ITERATION -- 2



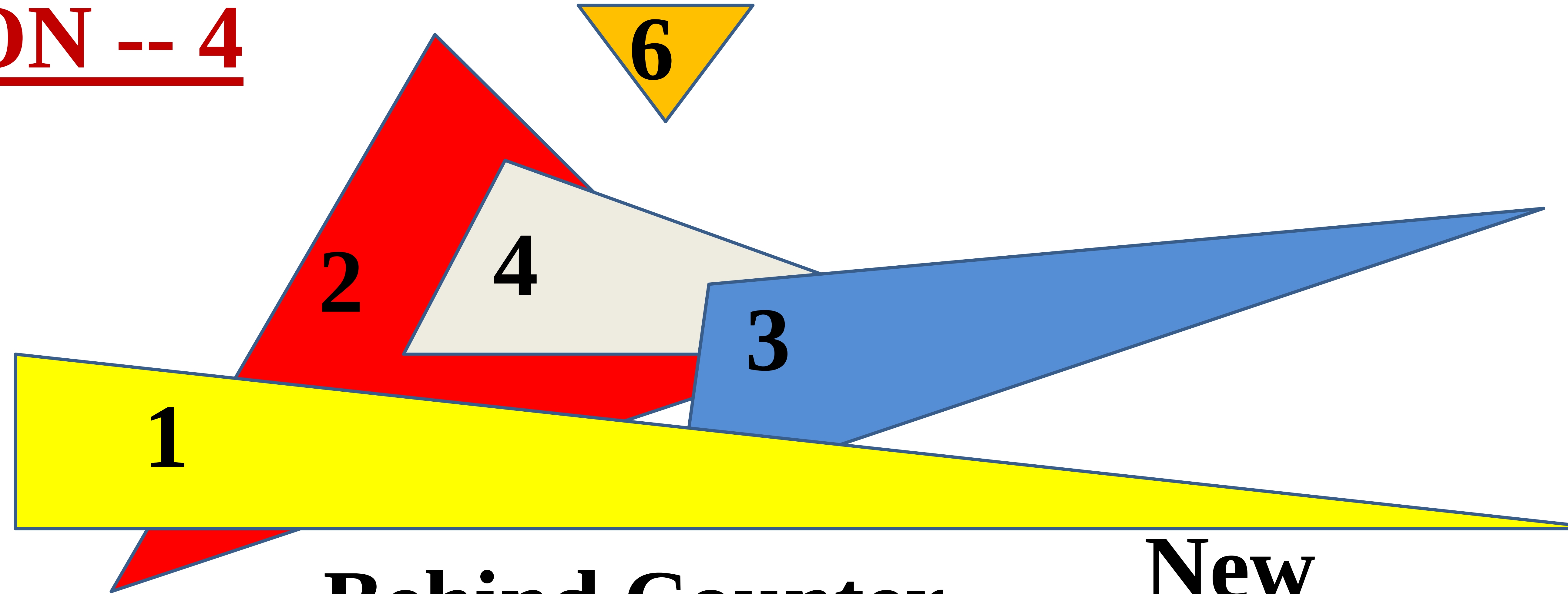
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	1	1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	1	0
4	3 ✖	0	-1
5	✖	2	2
6	✖	-1	-1

ITERATION -- 3



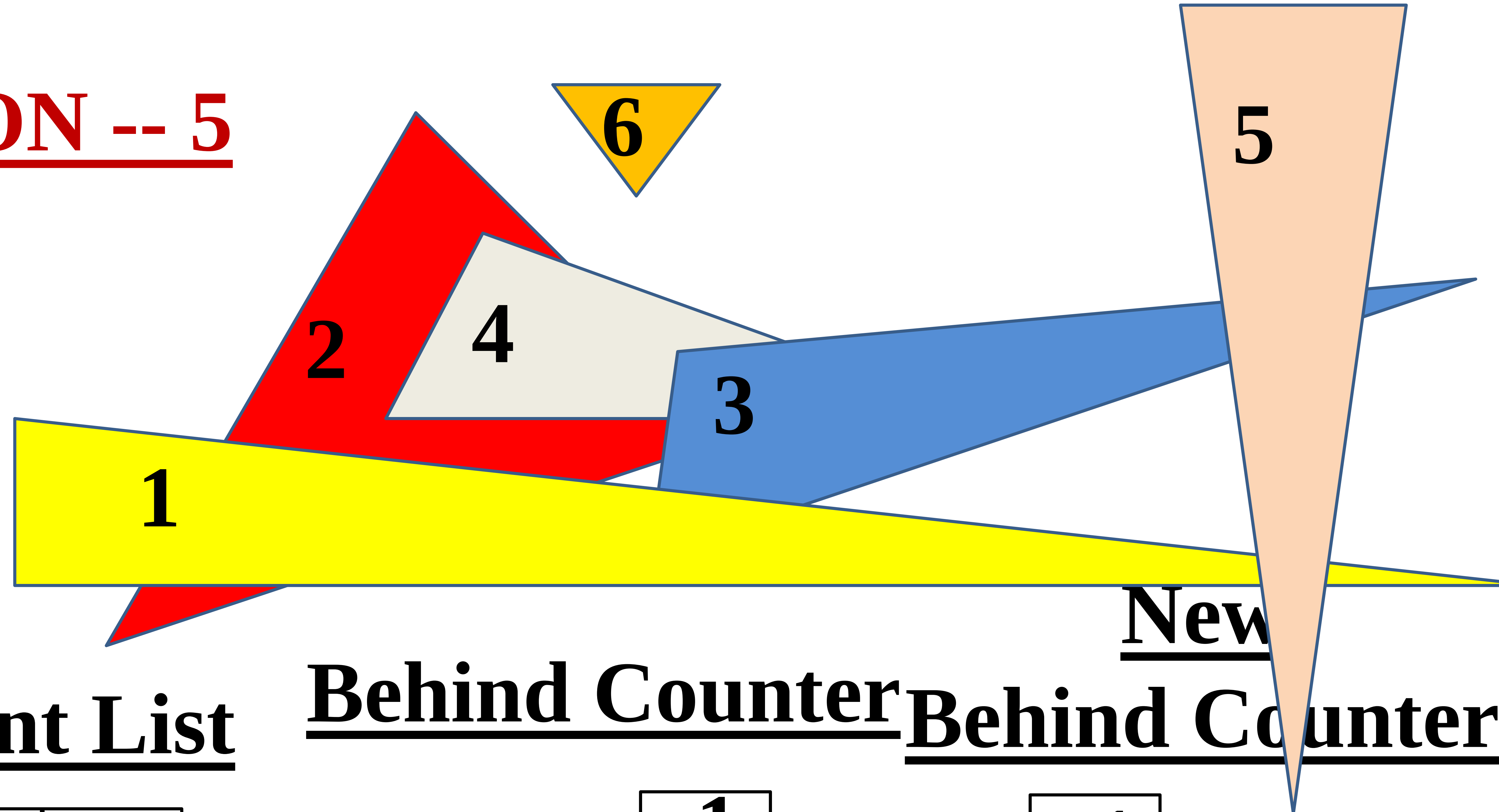
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	1	0
2	4 3 1 ✖	-1	-1
3	1 5 ✖	0	-1
4	3 ✖	-1	-1
5	✖	2	1
6	✖	-1	-1

ITERATION -- 4



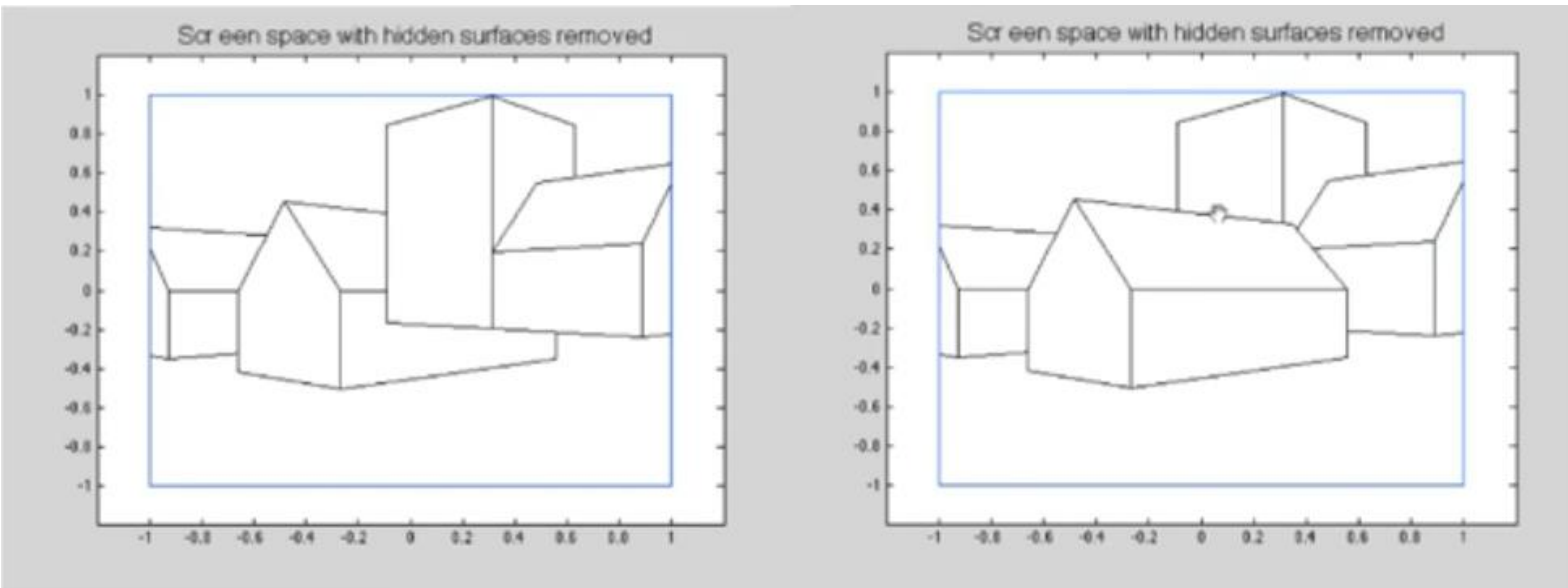
Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	0	-1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	-1	-1
4	3 ✖	-1	-1
5	✖	1	0
6	✖	-1	-1

ITERATION -- 5



Polygon Array	Front List	Behind Counter	New Behind Counter
1	5 ✖	-1	-1
2	4 3 1 ✖	-1	-1
3	1 5 ✖	-1	-1
4	3 ✖	-1	-1
5	✖	0	-1
6	✖	-1	-1

Before and After Painter's Algorithm



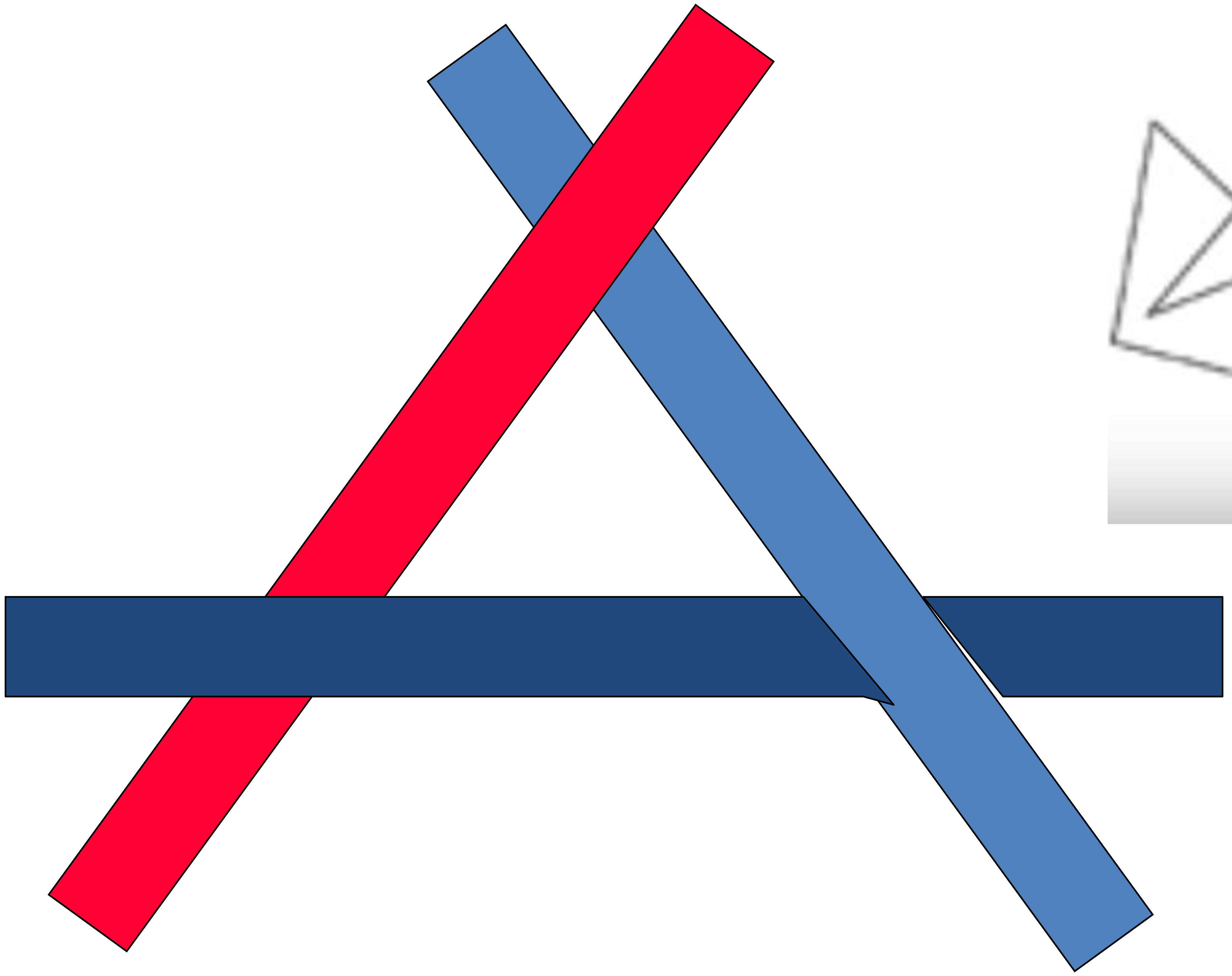
Painter’s Algorithm Limitations

- Requires sorting of polygons.
- All polygons must be necessarily filled.
- May lead to erroneous images if a failure condition occurs.

Painter's Algorithm

Failure Conditions

Polygons with cyclic overlap.



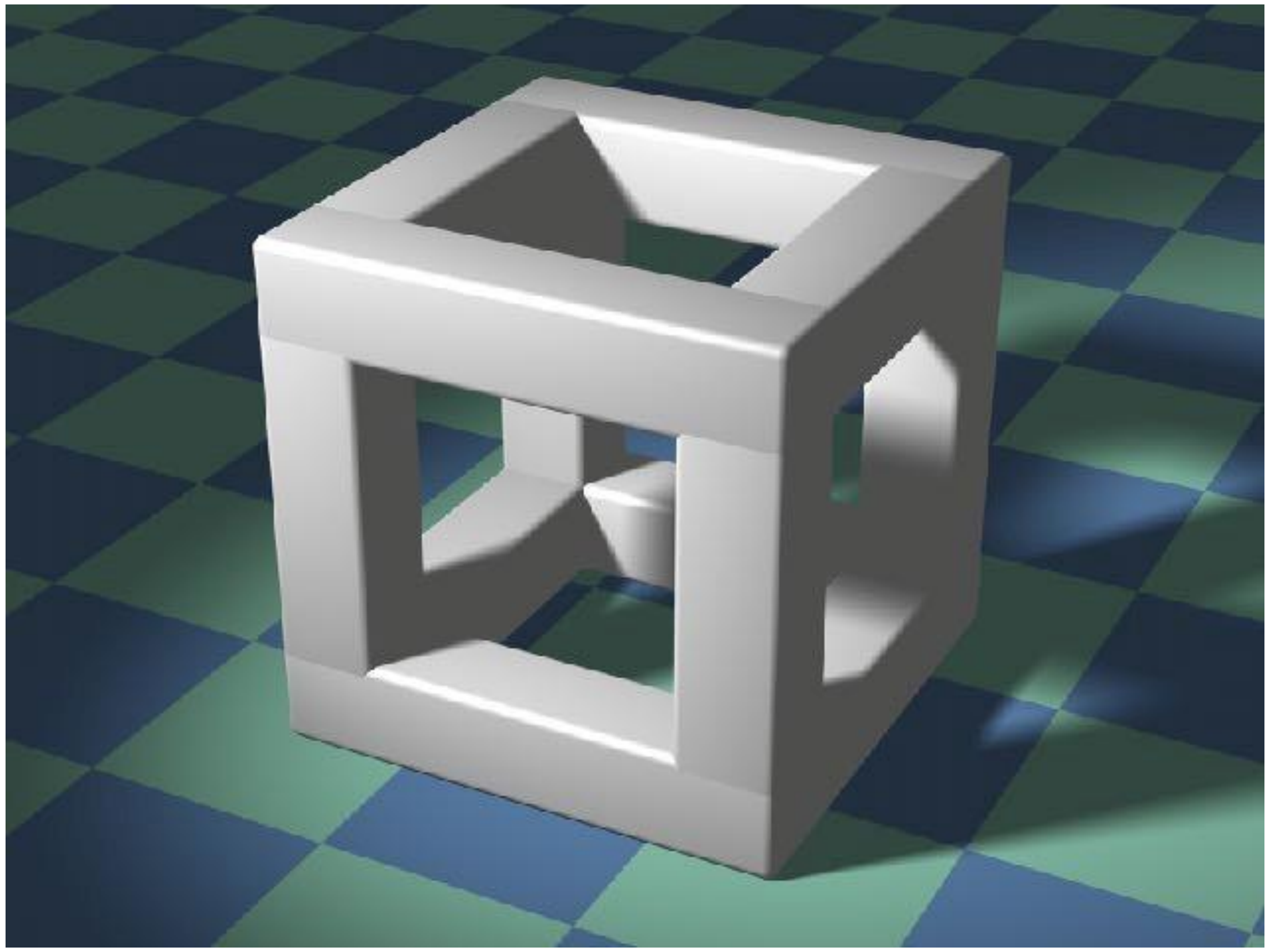
Z-Buffer

This is the hidden-surface-removal algorithm that eventually won.

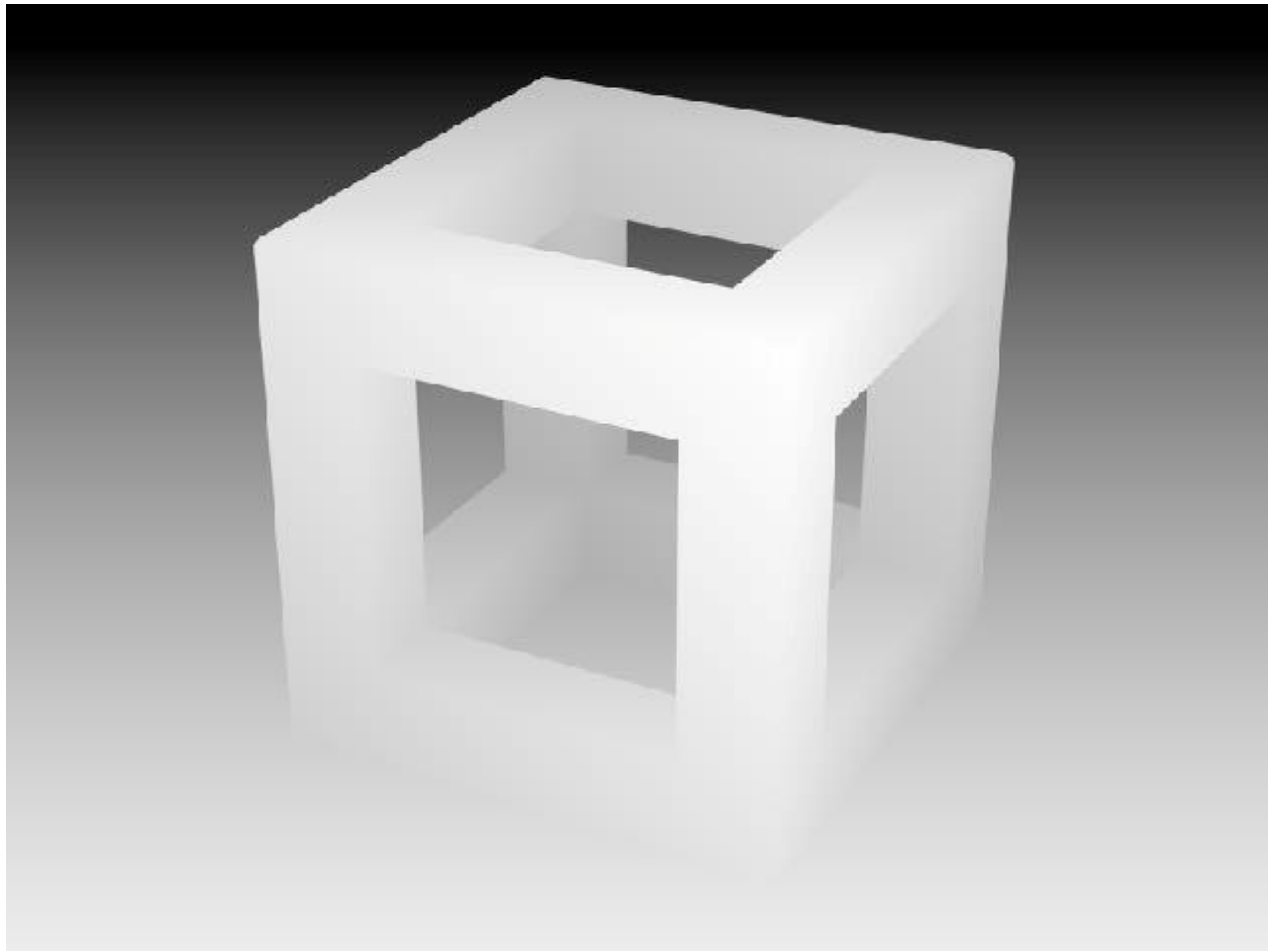
Idea:

- Store current min. z-value for <u>each</u> sample position
- Needs an additional buffer for depth values
- framebuffer stores RBG color values
- depth buffer (z-buffer) stores depth (16 to 32 bits)

Z-Buffer Example



Rendering



Depth buffer

Image credit: Dominic Alves, flickr.

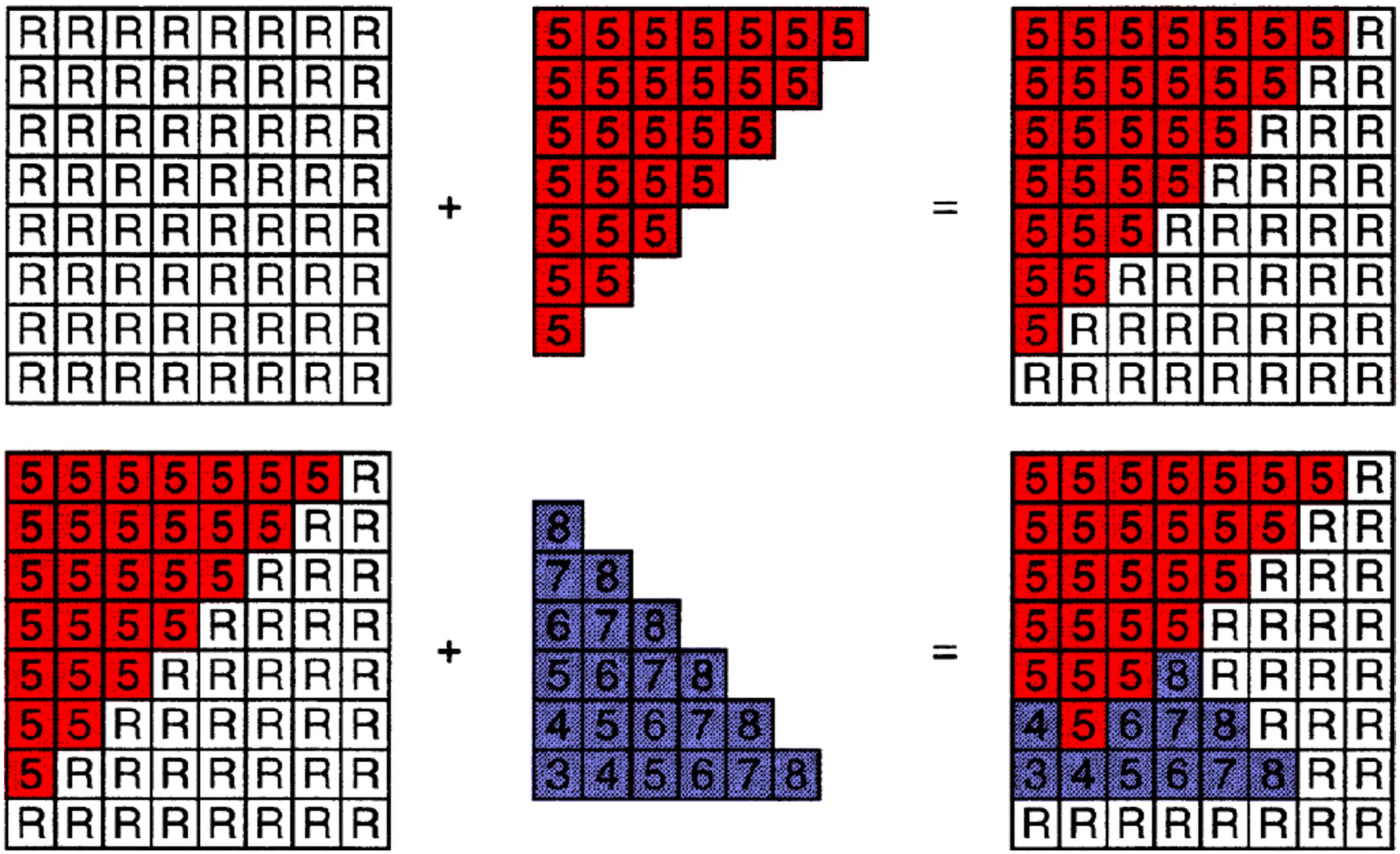
Z-Buffer Algorithm

Initialize depth buffer to ∞

During rasterization:

```
for (each triangle T)
    for (each sample (x,y,z) in T)
        if (z < zbuffer[x,y])              // closest sample so far
            framebuffer[x,y] = rgb;        // update color
            zbuffer[x,y]    = z;           // update z
        else
            ;                // do nothing, this simple is not closest
```

Z-Buffer Algorithm

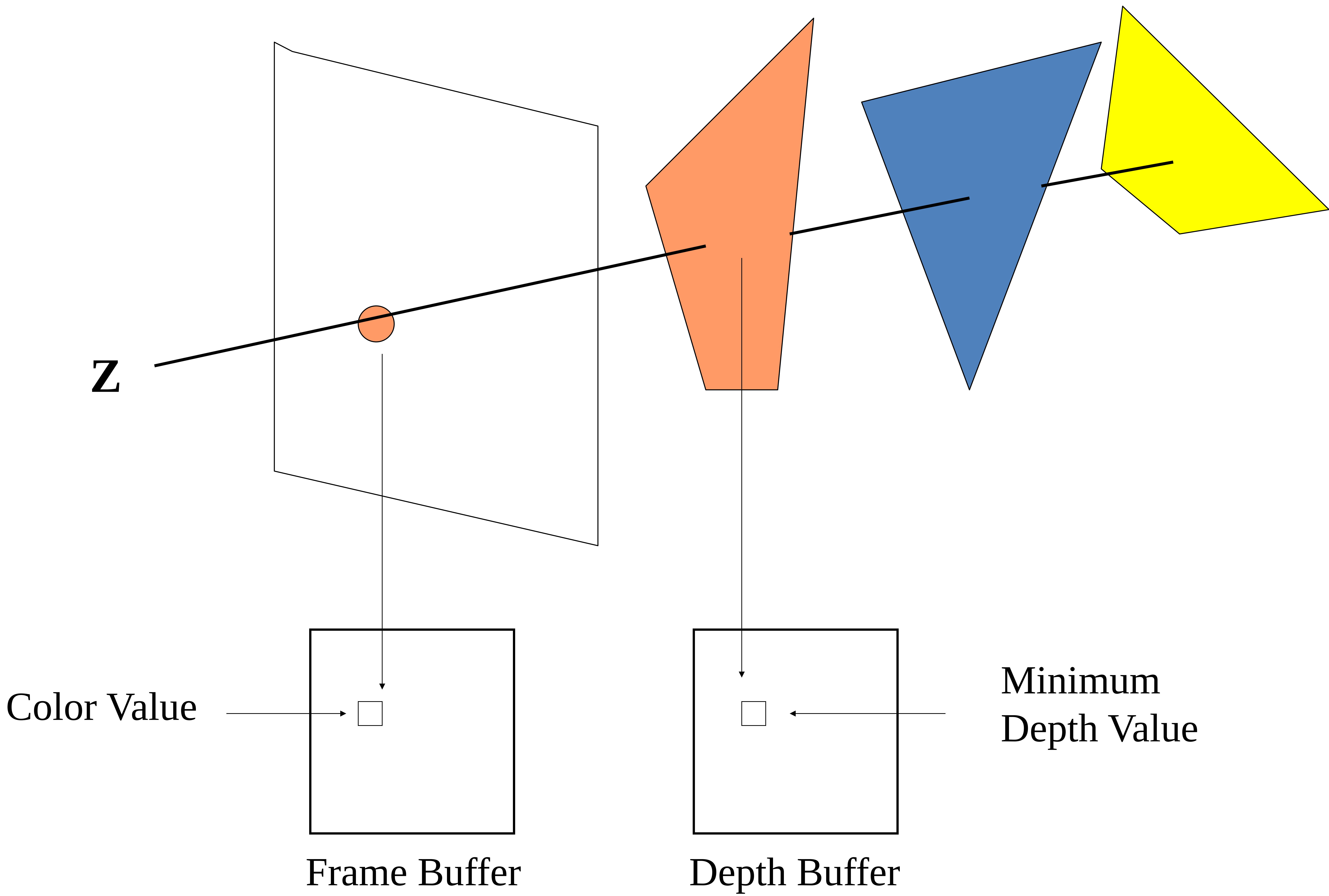


Z-Buffer Complexity

Complexity

- O(n) for n triangles
- Most important visibility algorithm
    - Implemented hardware for all GPUs
    - Used by OpenGL

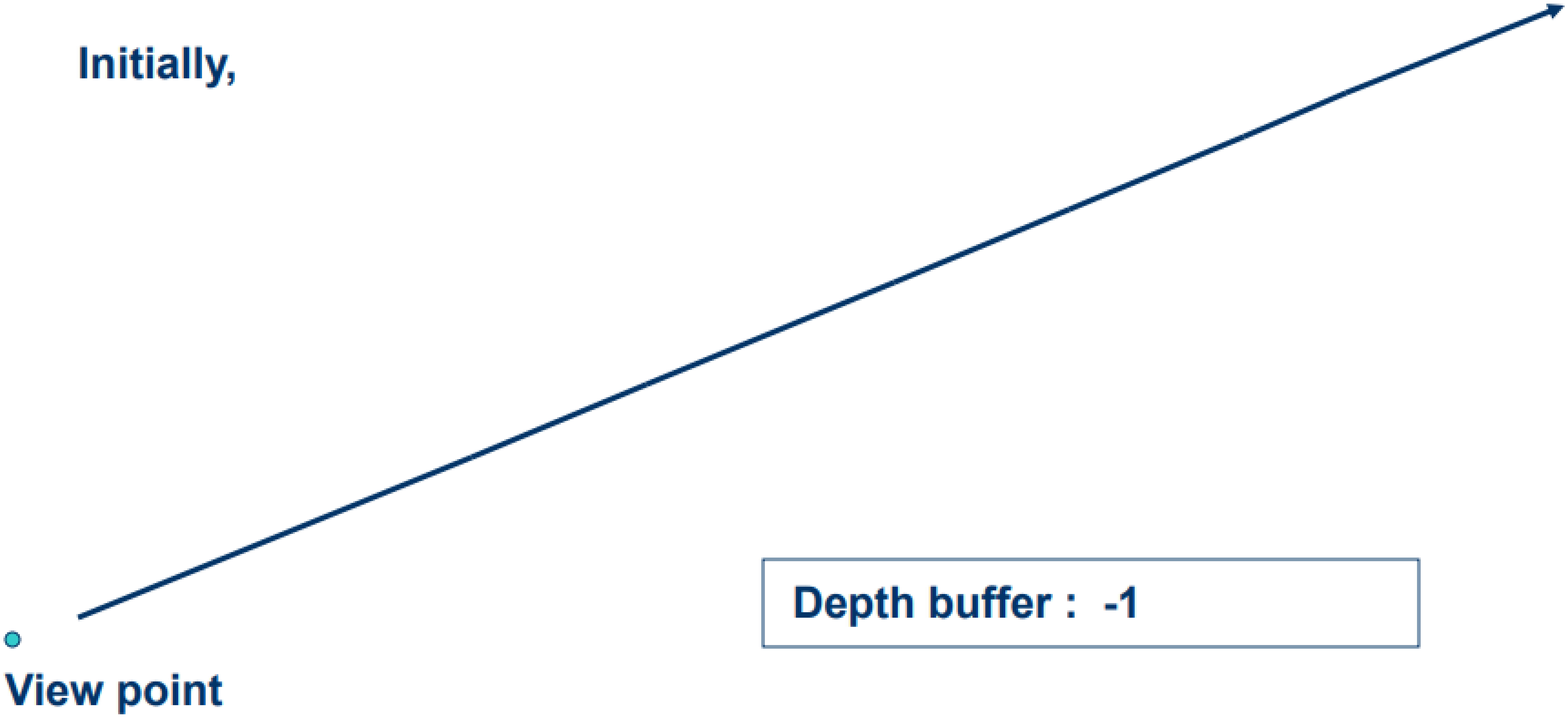
Z-Buffer


Z
Color Value
Minimum
Depth Value
Frame Buffer
Depth Buffer

Z-Buffer Algorithm

- For each pixel (*i*, *j*), a line passing through the pixel and the viewer is considered, and the depths of the polygons on this line are computed.
- The value *d*(*i*, *j*) in the depth buffer contains the pseudo-depth of the closest polygon encountered at pixel (*i*, *j*).
- The value *p*(*i*, *j*) in the frame buffer (the color of the pixel) is the color of the closest polygon.

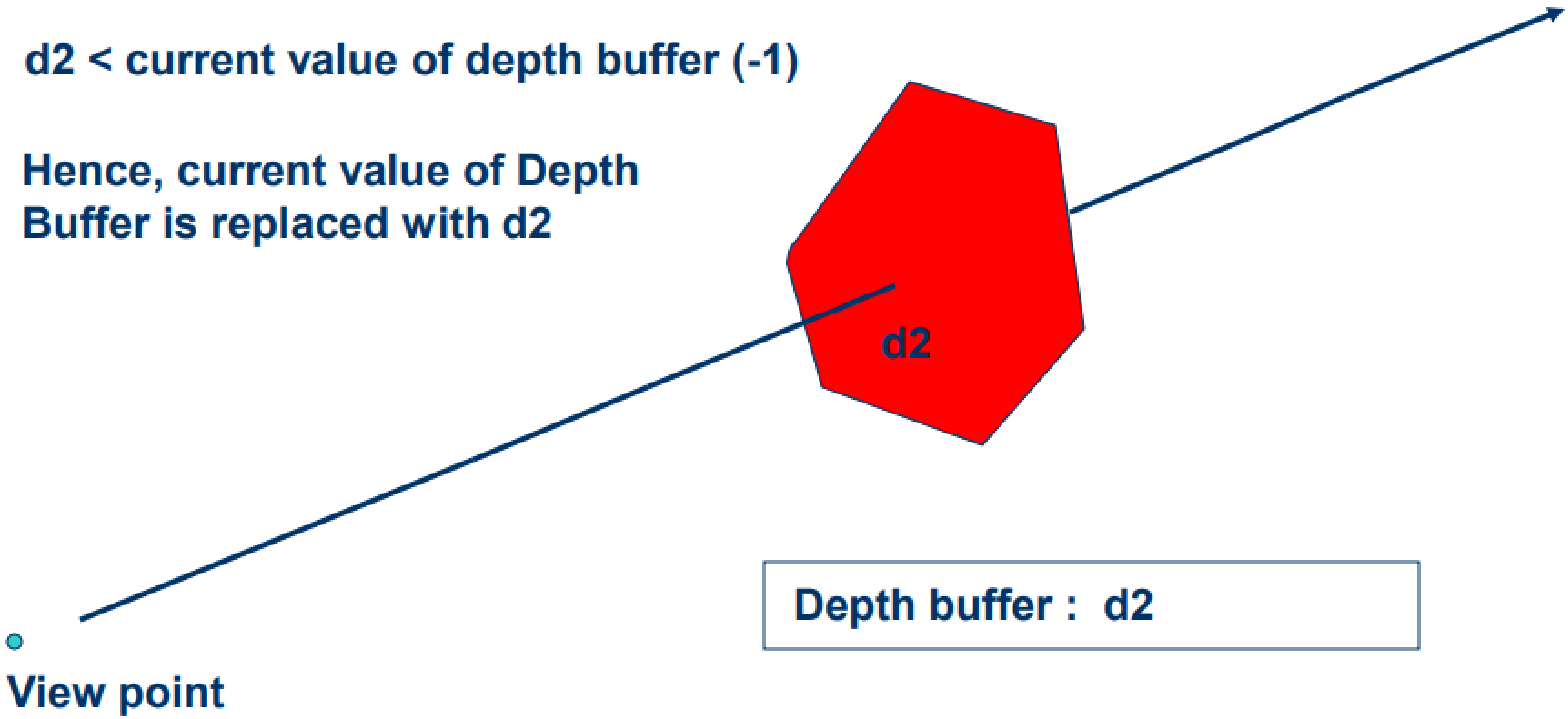
Z-Buffer



Z-Buffer

d2 < current value of depth buffer (-1)

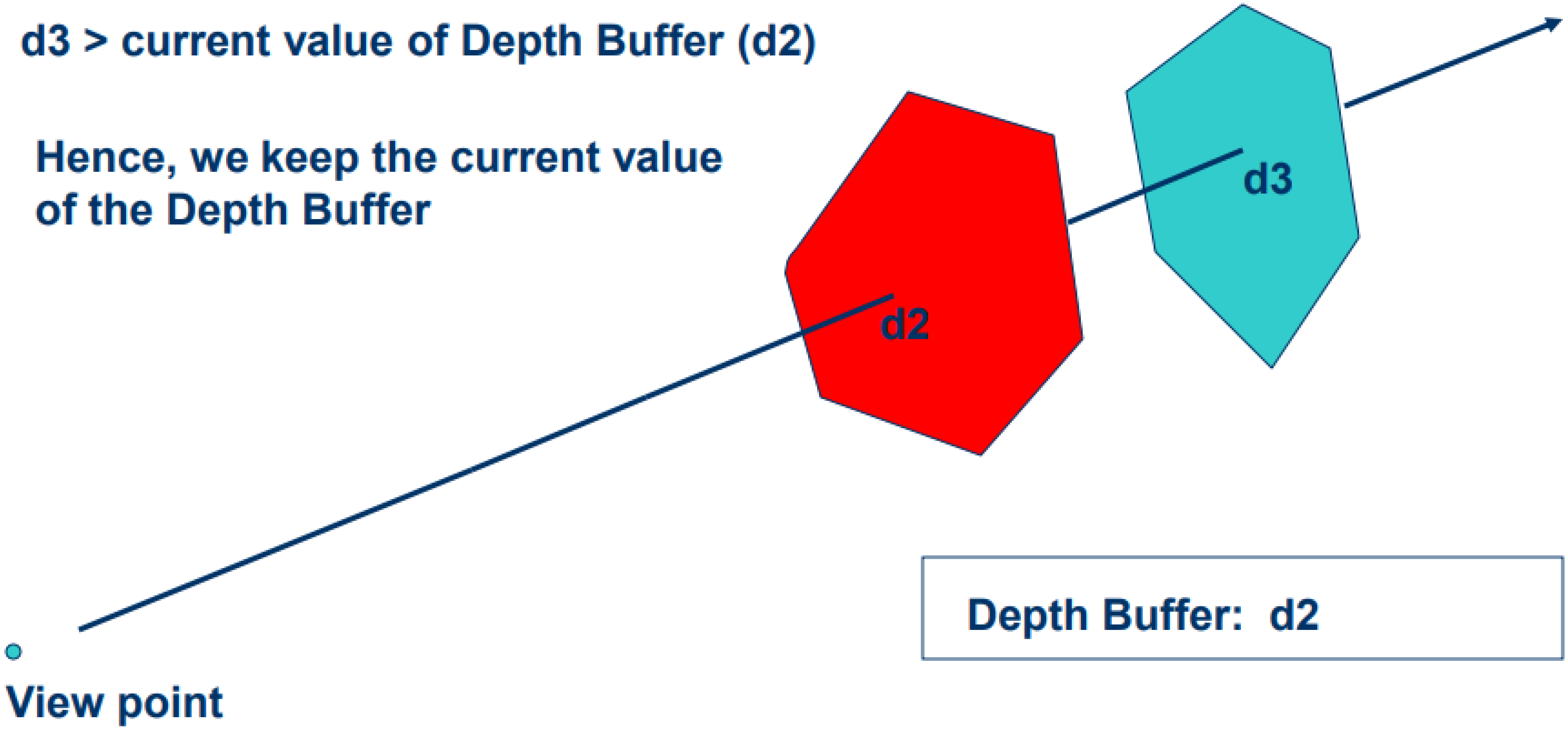
Hence, current value of Depth Buffer is replaced with d2



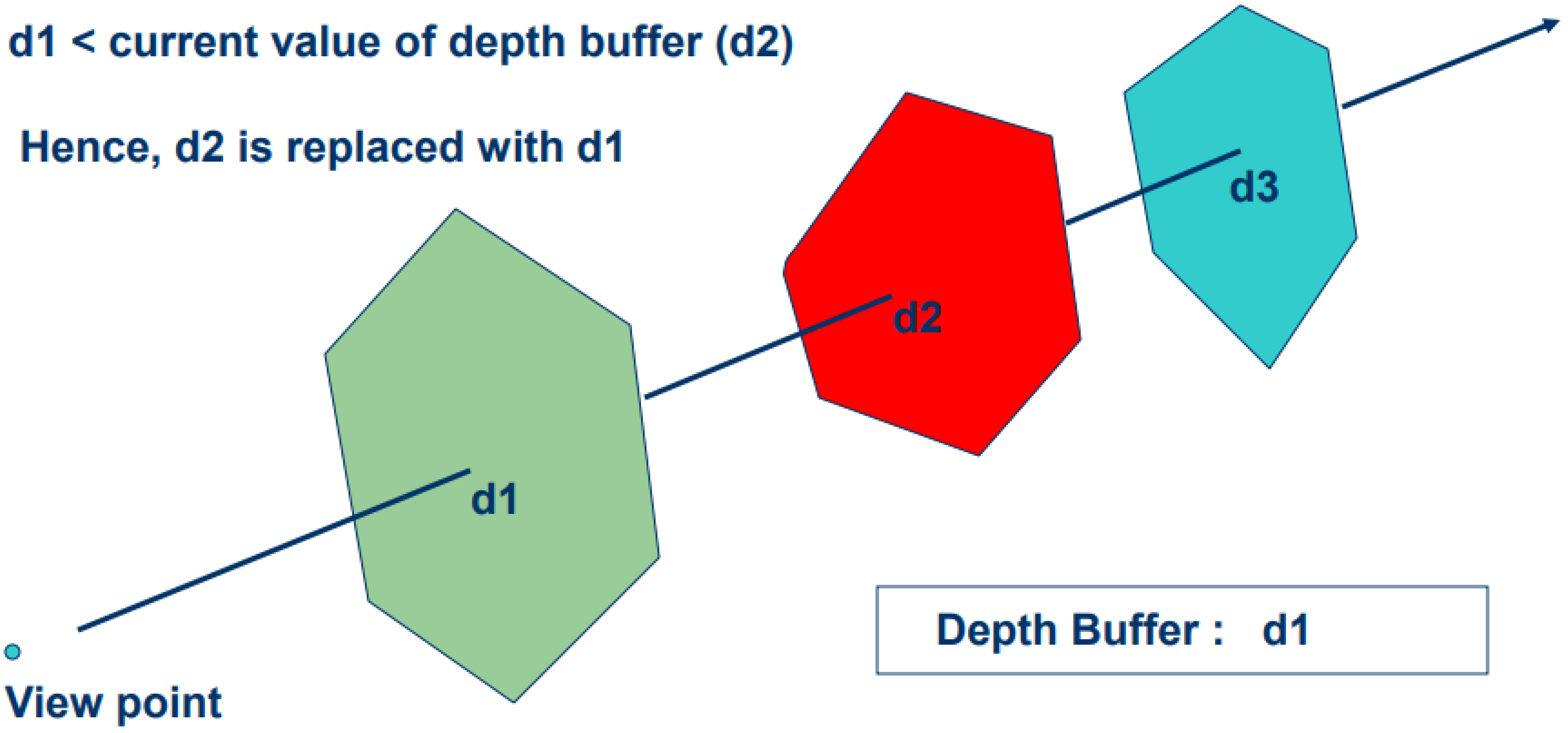
Z-Buffer

d3 > current value of Depth Buffer (d2)

Hence, we keep the current value of the Depth Buffer



Z-Buffer



d1 < current value of depth buffer (d2)
Hence, d2 is replaced with d1
d3
d2
d1
Depth Buffer : d1
View point

Z-Buffer Limitations

– The algorithm requires a large amount of additional memory to store the pseudo depth at each pixel value.

Z-Buffer Example

- For a 1280 x 1024 pixel display using 16Gb color model and 24b depth buffer, the memory required is?

Memory= depth buffer +frame buffer

=(1280x1024x24)+ (1280x1024x16)

=31,457,280+20,971,520

=52,428,800b

=6,553,600B