Computer Graphics

13 – Scan Conversion, Visibility

Yoonsang Lee
Hanyang University

Spring 2023

Final Exam Announcement (same as before)

- Date & time: **June 12 (Mon), 7:30 - 8:30 PM**
- Place: IT.BT **507, 508**
  - Student list for each room will be announced soon.

- Scope: Lecture & Lab **8 ~ 13**

- **You cannot leave until 30 minutes after the start of the exam** even if you finish the exam earlier.

- That means, **you cannot enter the room after 30 minutes from the start of the exam** (do not be late, never too late!).

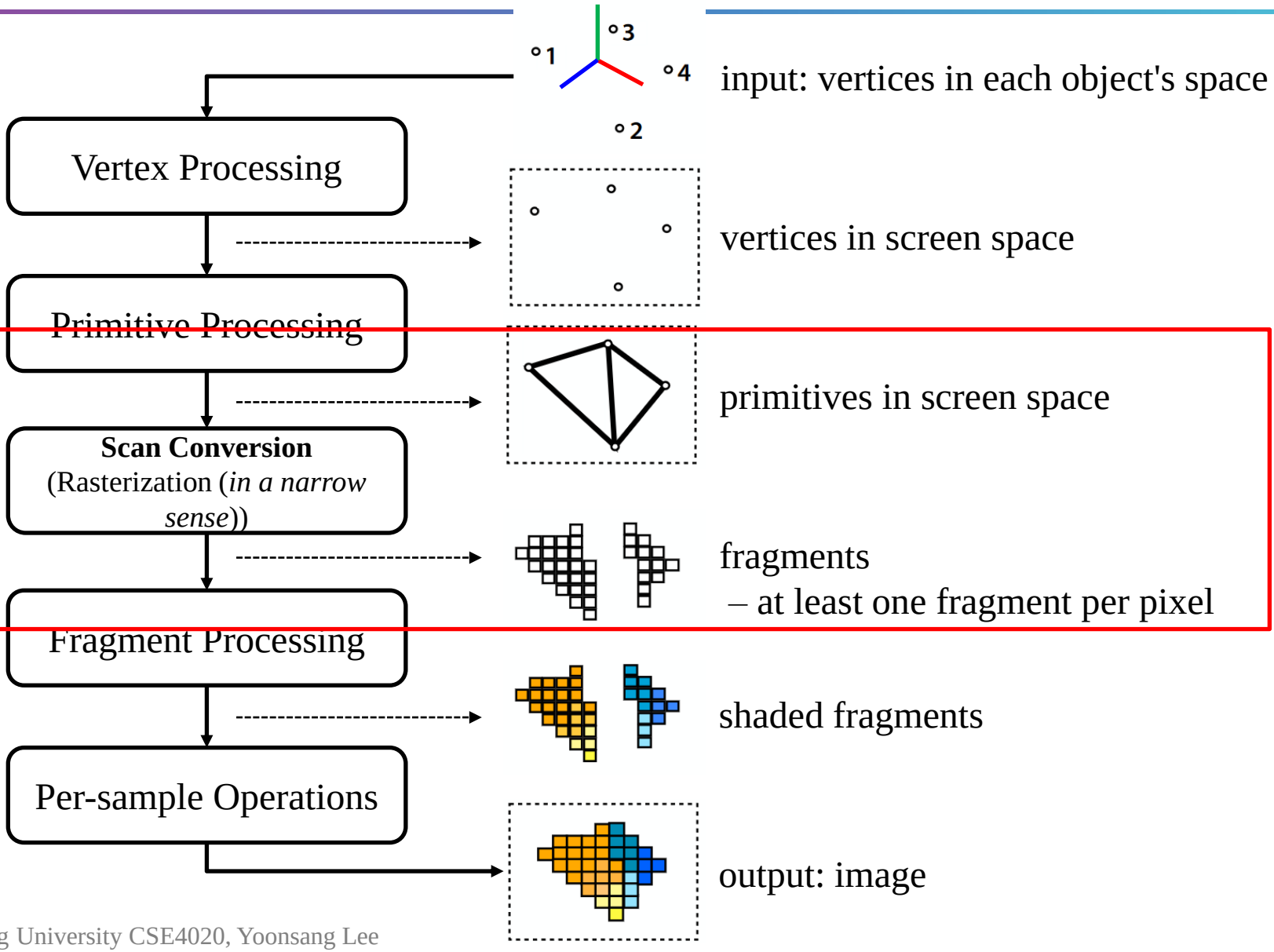
- Please bring your **student ID card** to the exam.

Outline

- Scan Conversion
- Visibility Problem
  - Clipping (Viewing frustum culling)
  - Back-face culling
  - Hidden surface removal
- Rendering Pipeline Again
- Course Wrap-up

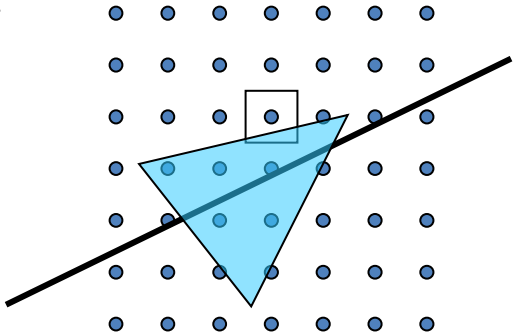
Scan Conversion

Recall: Rendering Pipeline



Scan Conversion

- Scan conversion process converts vertex representation to pixel representation (fragments).



- First job: Determine which fragments belong to a primitive.

- Second job: Interpolate values across the primitive.
  - e.g., interpolated colors / normals / texture coordinates, …

Scan Conversion

- Algorithms for finding fragments for a primitive are called "drawing" algorithms.

- A primitive refers to basic geometric shapes such as points, lines, circles, and polygons.

- Line drawing algorithms
  - Digital differential analyzer (DDA)
  - Bresenham's line algorithm (1962)
  - Xiaolin Wu's line algorithm(1991)
  - …
    - For details, refer to https://www.tutorialspoint.com/computer_graphics/line_generation_algorithm.htm

Scan Conversion

- Circle drawing algorithms
  - Midpoint circle algorithm
  - Bresenham's circle algorithm
  - Xiaolin Wu's circle algorithm
  - ...
    - For details, refer to https://www.tutorialspoint.com/computer_graphics/circle_generation_algorithm.htm

- Polygon drawing algorithms
  - Scanline
  - Boundary fill
  - Flood fill
  - …
    - For details, refer to https://www.tutorialspoint.com/computer_graphics/polygon_filling_algorithm.htm

Scan Conversion

- We'll just skip details of these algorithms.

- Actually, these tasks are not so easy as one might think.
  - Computational efficiency, anti-aliasing, ...

- But most graphics APIs (including OpenGL) basically support these operations.
  - These algorithms were intensively studied in early days of computer graphics, so quite mature now.
  - Now these algorithms are implemented in graphics hardware (GPU).

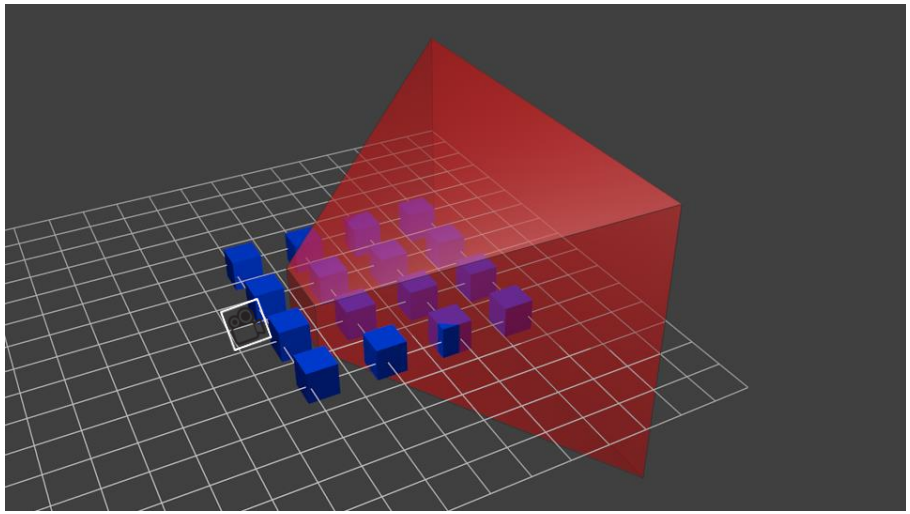
- So nowadays you can think that you can simply draw them by making use of graphics APIs.

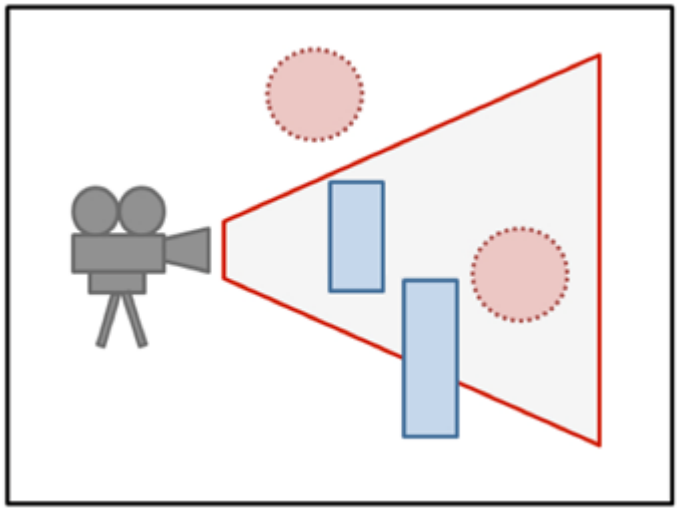
Visibility Problem

Visibility Problem

- What is VISIBLE?

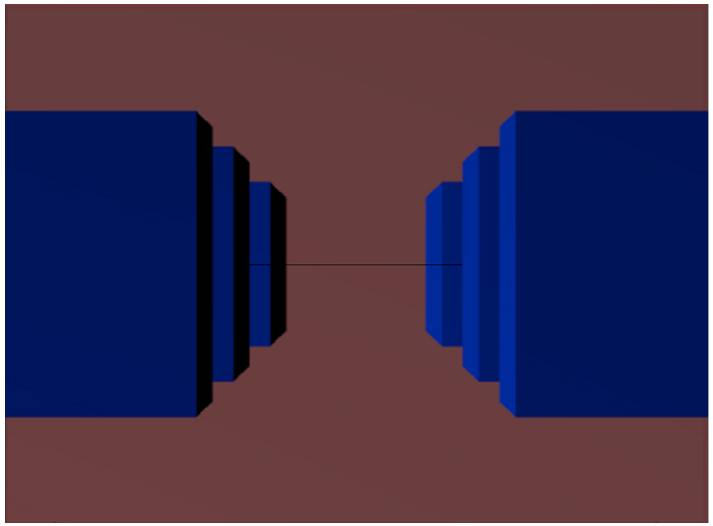
Red: viewing frustum, Blue: objects

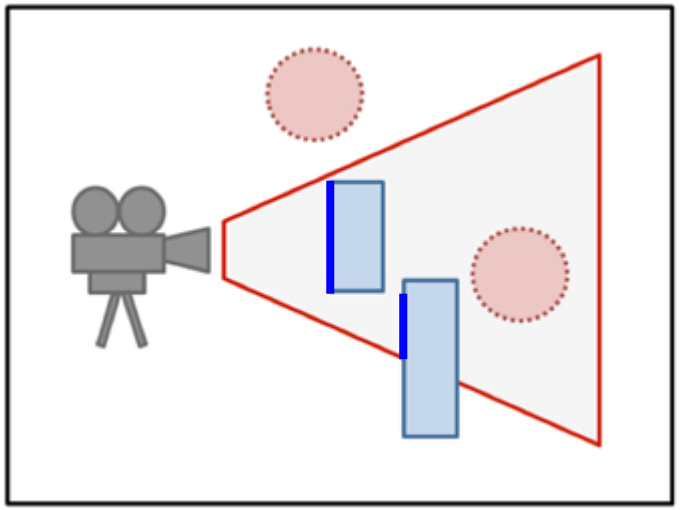




Visibility Problem

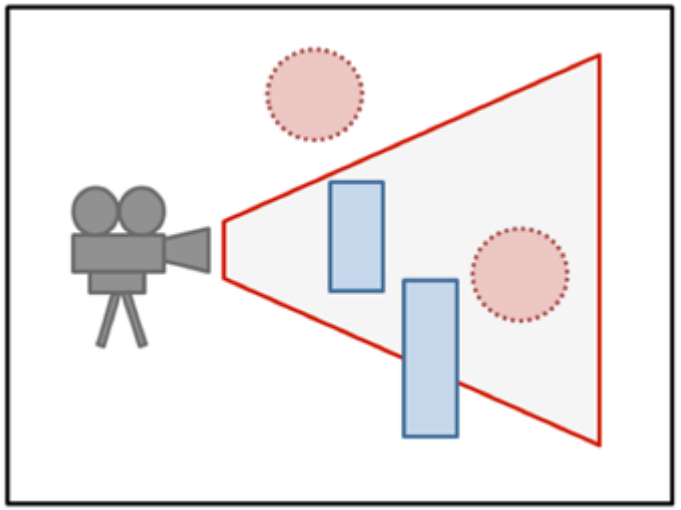
- The answer is:





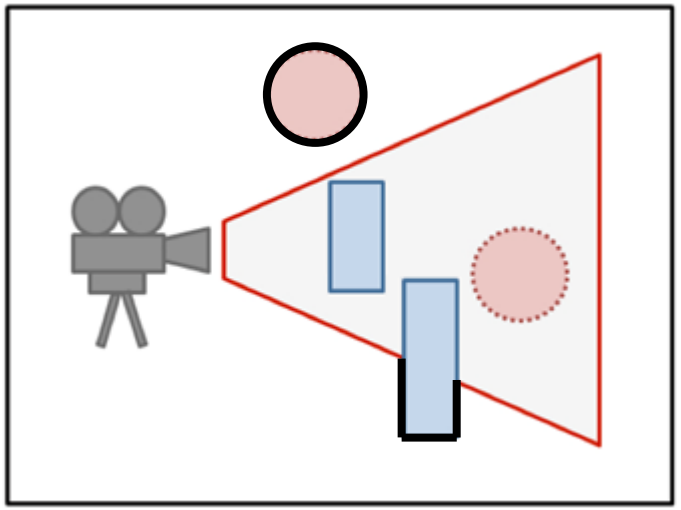
Visibility Problem

- What is NOT VISIBLE?



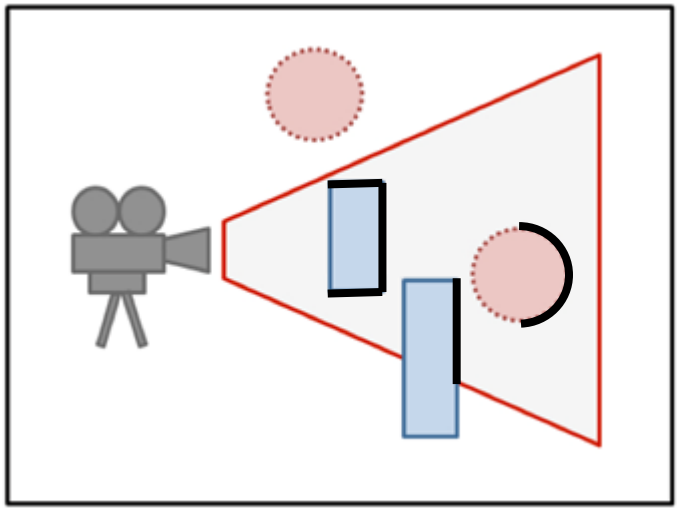
Visibility Problem

- What is NOT VISIBLE?
- **Primitives outside the viewing frustum**



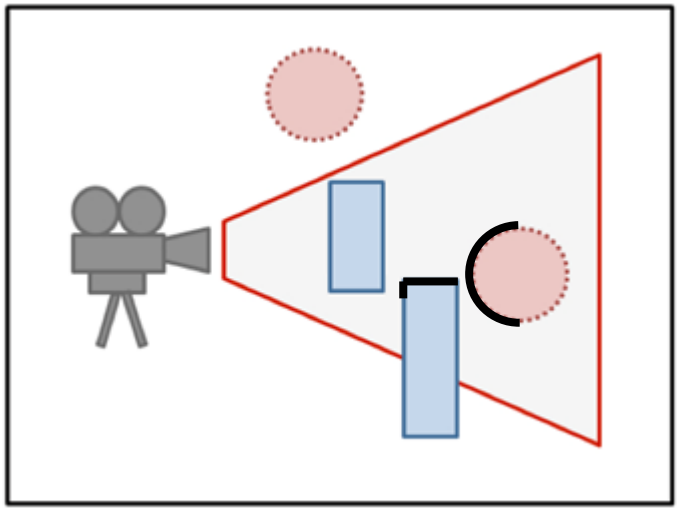
Visibility Problem

- What is NOT VISIBLE?
- Primitives outside the viewing frustum
- **Back-facing primitives**



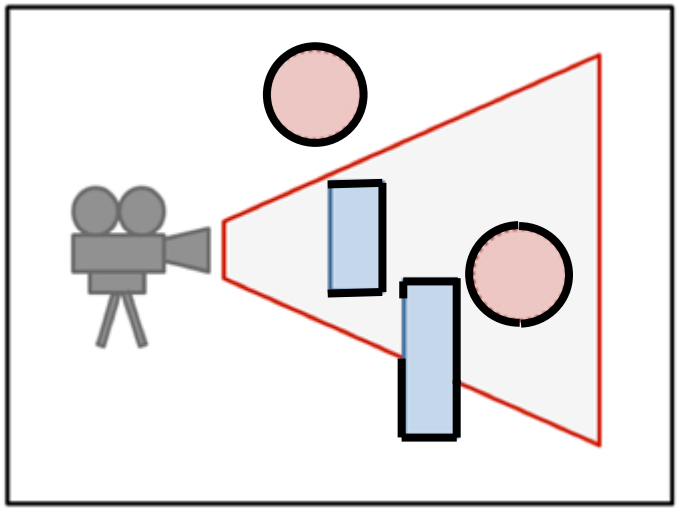
Visibility Problem

- What is NOT VISIBLE?
- Primitives outside the viewing frustum
- Back-facing primitives
- **Primitives occluded by other objects closer to the camera**



Visibility Problem

- These **invisible primitives should be removed** because…
- No need to spend time to process invisible vertices and polygons.
- A close object must hide a farther one.
- So, removing these primitives is required for efficient and correct rendering.



Visibility Problem

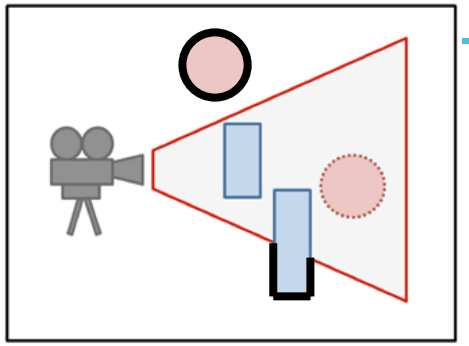
- Removing...

- Primitives outside the viewing frustum
- → **Clipping (Viewing frustum culling)**

- Back-facing primitives
- → **Back-face culling**

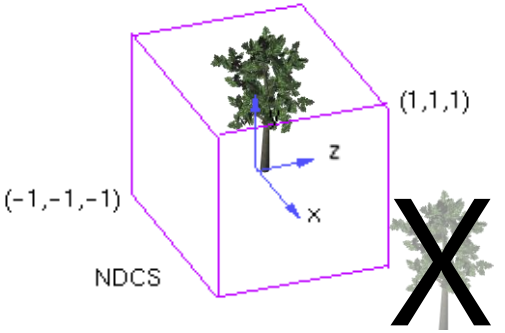
- Primitives occluded by other objects closer to the camera
- → **Hidden surface removal**

Clipping (Viewing Frustum Culling)

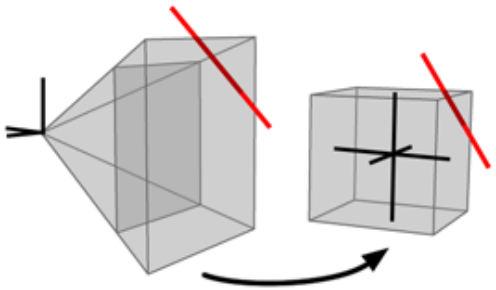


- Removing primitives outside the viewing frustum

- Clipping is performed in clip space.
  - Recall: A point (x',y',z') in NDC space remains unclipped if it's in canonical view volume (== if $-1 \le x',y,'z' \le 1$).
  - A point (x,y,z,w) in **clip space** remains unclipped if $-w \le x,y,z \le w$.
    - By clipping before perspective division (in clip space), it saves time by not computing perspective division for the clipped primitives.
  - Computation is much simpler than view space.
  - That's why the space's name is "clip space".

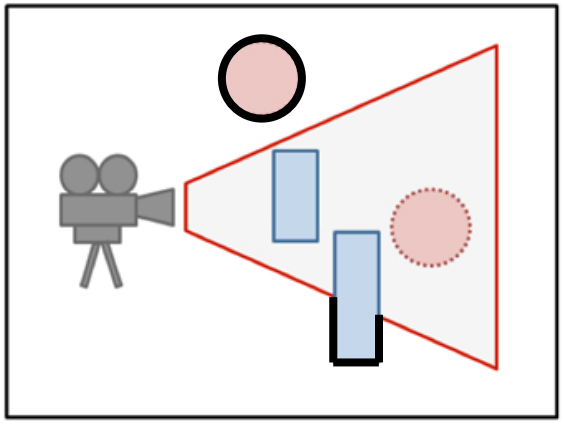


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$



Clipping (Viewing Frustum Culling)

- Line clipping algorithms
  - Cohen–Sutherland (1967)
  - Cyrus–Beck (1978)
  - Liang–Barsky (1984)
  - ...

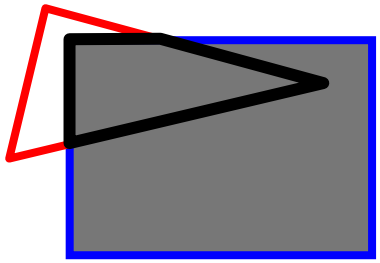


- Polygon clipping algorithms
  - Sutherland–Hodgman (1974)
  - Weiler–Atherton (1977)
  - ...

    - For details, refer to https://www.tutorialspoint.com/computer_graphics/viewing_and_clipping.htm

Clipping (Viewing Frustum Culling)

- Polygon clipping algorithms are more complicated.
  - Vertices may be added to or deleted from the triangle.



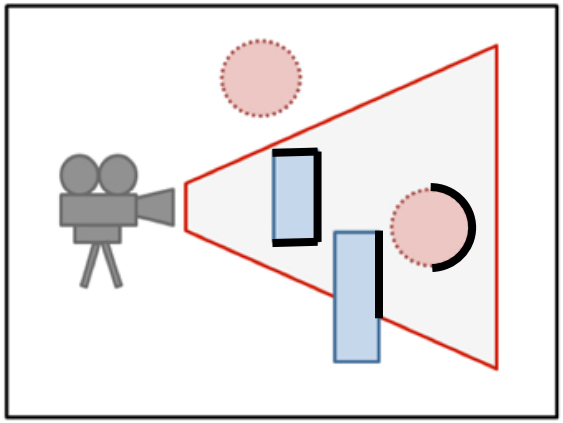
triangle → quad

- Again, let's just skip details of these algorithms.

- Most graphics APIs (including OpenGL) performs clipping by default.
  - You just set the view frustum, then OpenGL will do clipping for you.

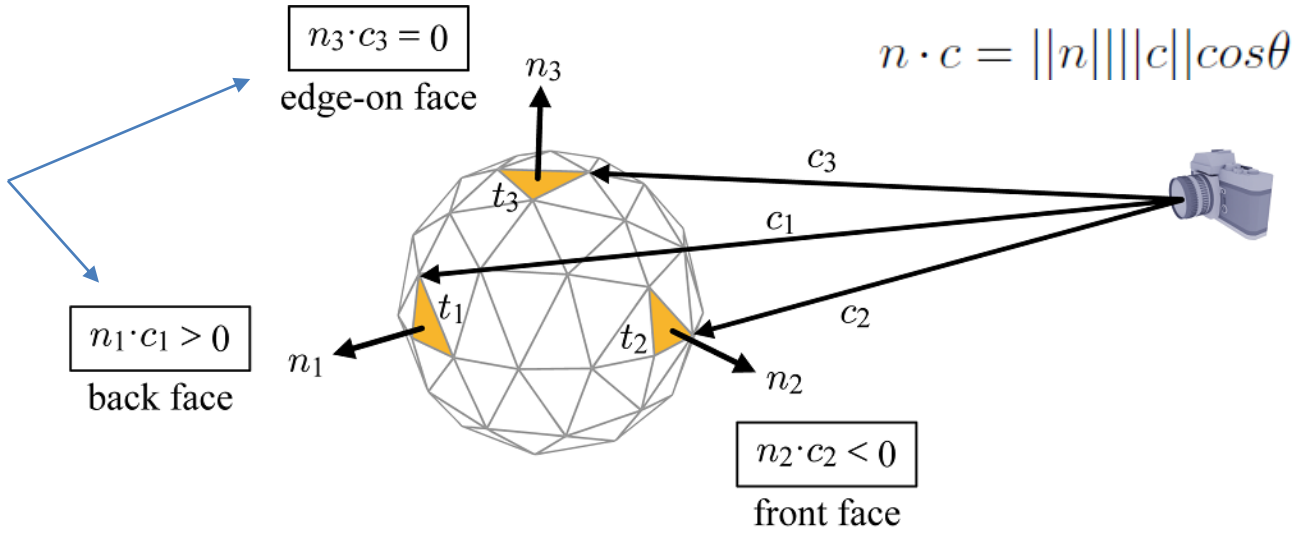
Back-Face Culling

- Removing back-facing primitives



- Determined by the dot product of normal and view (camera) vectors.

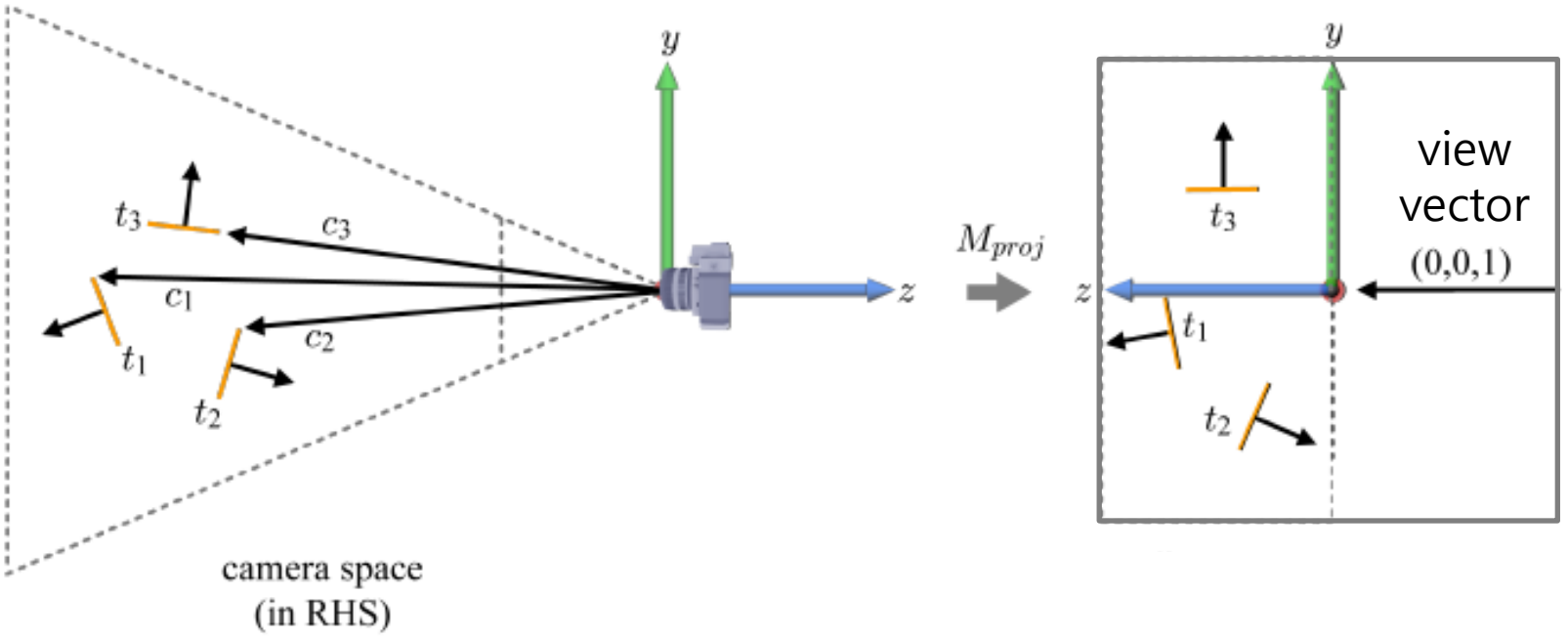
Discard!



* This image is from the slides of Prof. JungHyun Han (Korea Univ.): http://media.korea.ac.kr/book/

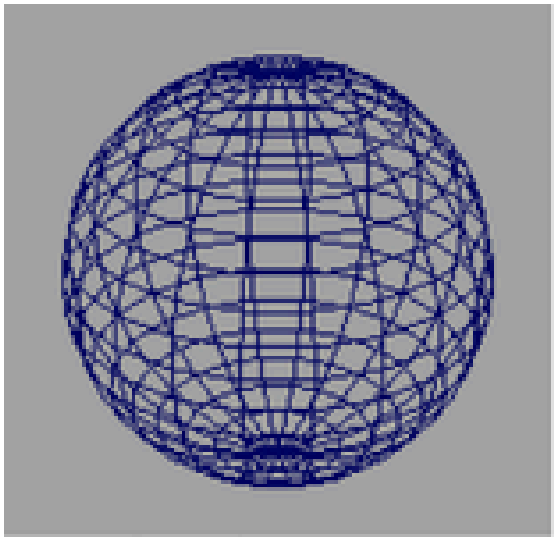
Back-Face Culling

- Back-face culling is performed in NDC space.
  - Because in NDC space, we can use a single view vector, (0,0,1), which is much more efficient.

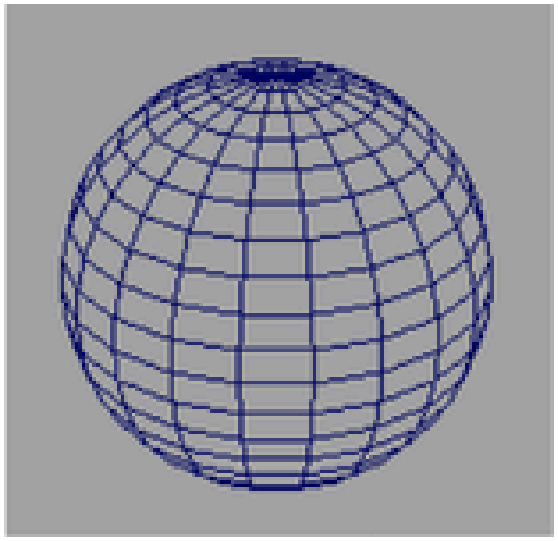


* This image is from the slides of Prof. JungHyun Han (Korea Univ.): http://media.korea.ac.kr/book/

Back-Face Culling



Backfaces

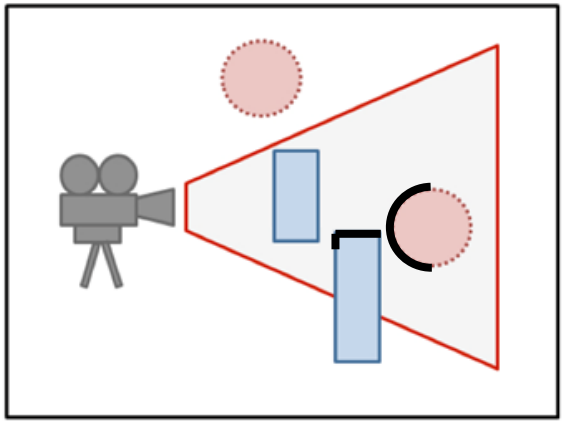


No backfaces

* This image is from https://help.autodesk.com/view/MAYAUL/2024/ENU/?guid=GUID-B7F70ACE-0F3F-483B-83B5-D9711D6CBAAC

Hidden Surface Removal

- Removing primitives occluded by other objects closer to the camera



- Also known as
  - Hidden Surface Elimination
  - Hidden Surface Determination
  - Visible Surface Determination
  - Occlusion Culling

Hidden Surface Removal

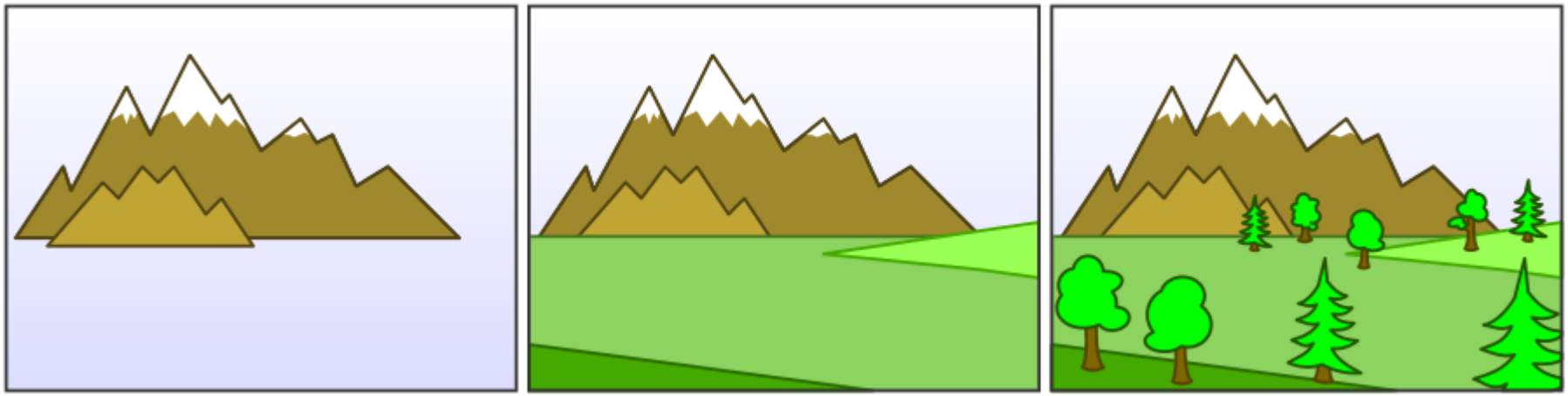
- Many algorithms
  - Z-buffering (a.k.a. Depth buffering)
  - Painter's algorithm
  - BSP tree
  - ...

- Z-buffering is the standard method.

- Let's see the ideas of Painter's algorithm & Z-buffering.

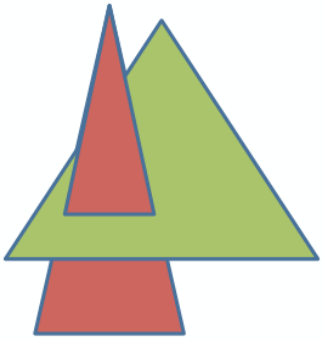
Painter's Algorithm

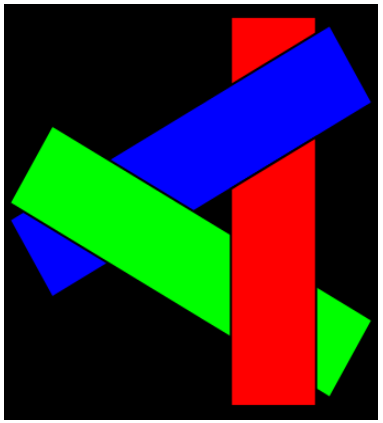
- Sorts all polygons based on their distance from the viewer, or *depth*, and then paints them in this order, farthest to closest.
- Polygons that are closer to the viewer will be drawn on top of polygons that are farther away.
- Works on a polygon-by-polygon basis.



Weakness of Painter's Algorithm

- What if there are cycles in the sorted graph?
  - The only solution is dividing these polygons into small pieces.





- Requires sorting all polygons by their depth whenever the viewer's perspective or object placement changes.

- → Time-consuming!

Z Buffering

- Maintain a separate buffer called the *z-buffer* (a.k.a. *depth buffer*), which stores the depth of each pixel on the screen.

- During the rendering process, for each pixel being processed, the z-buffer is checked to determine if the new object being rendered is closer to the viewer than the existing object.
  - If it is closer, the new object is drawn, and its depth value is updated in the z-buffer.
  - If it is farther away, the new object is discarded, and the existing object remains visible.

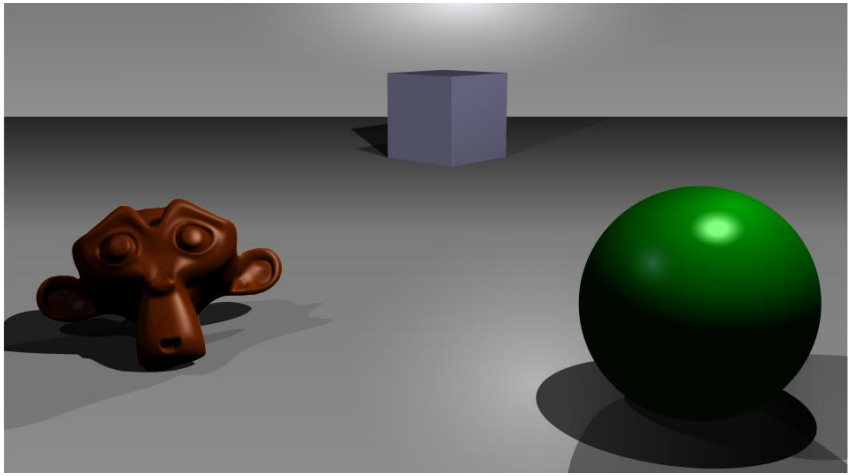
- Works on a pixel-by-pixel basis.

Z-Buffering: Algorithm

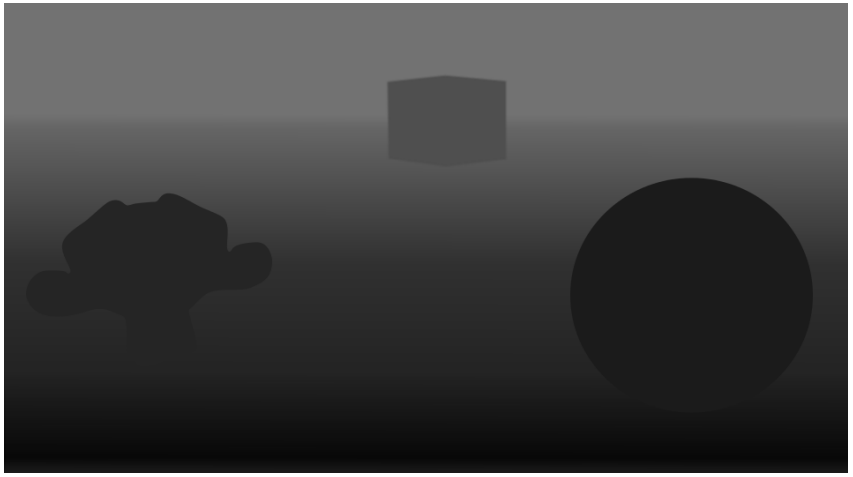
```
allocate depth_buffer;          // Allocate depth buffer → Same size as viewport.

for each pixel (x,y)            // For each pixel in viewport.
  write_frame_buffer(x,y,backgrnd_color);   // Initialize color.
  write_depth_buffer(x,y,farPlane_depth);   // Initialize depth (z) buffer.

for each polygon                // Draw each polygon (in any order).
  for each pixel (x,y) in polygon   // Rasterize polygon.
    color = polygon's color at (x,y);
    p_z = polygon's z-value at (x,y);// Interpolate z-value at (x, y).
    if (p_z < read_depth_buffer(x,y))       // If new depth is closer:
      write_frame_buffer(x,y,color);        //  Write new (polygon) color.
      write_depth_buffer(x,y,p_z);          //  Write new depth.
```

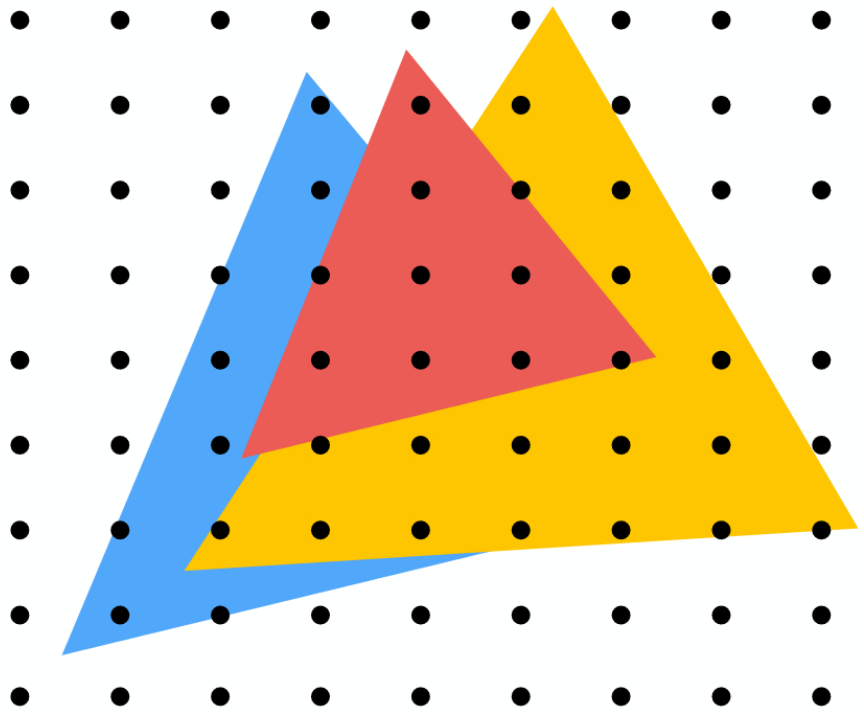


Frame buffer



Z-buffer (Depth buffer)

Example: rendering three opaque triangles



* The following slide is from the slides of Prof. Kayvon Fatahalian and Prof. Keenan Crane (CMU):
http://15462.courses.cs.cmu.edu/fall2015/

Occlusion using the depth-buffer (Z-buffer)

Processing yellow triangle:

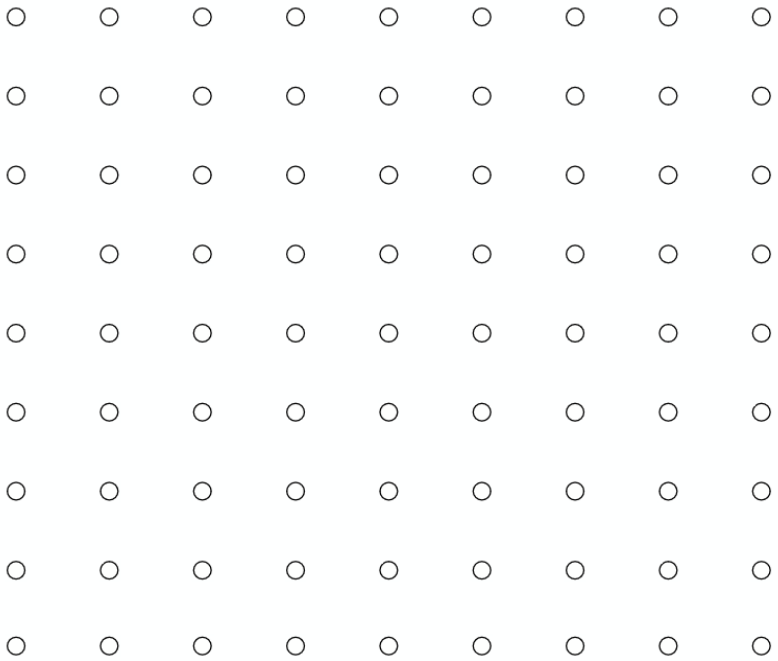
depth = 0.5

Grayscale value of sample point used to indicate distance

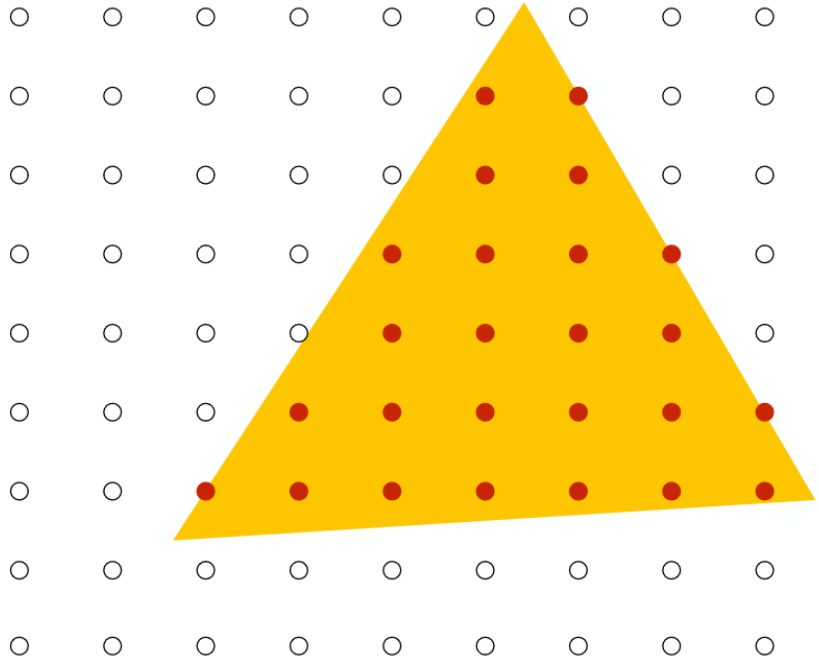
White = large distance

Black = small distance

Red = sample passed depth test



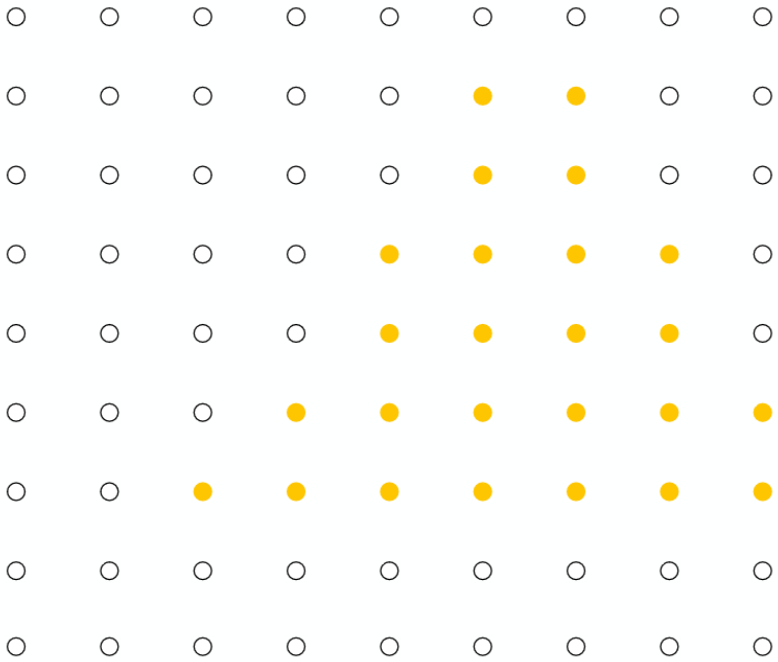
Color buffer contents



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing yellow triangle:



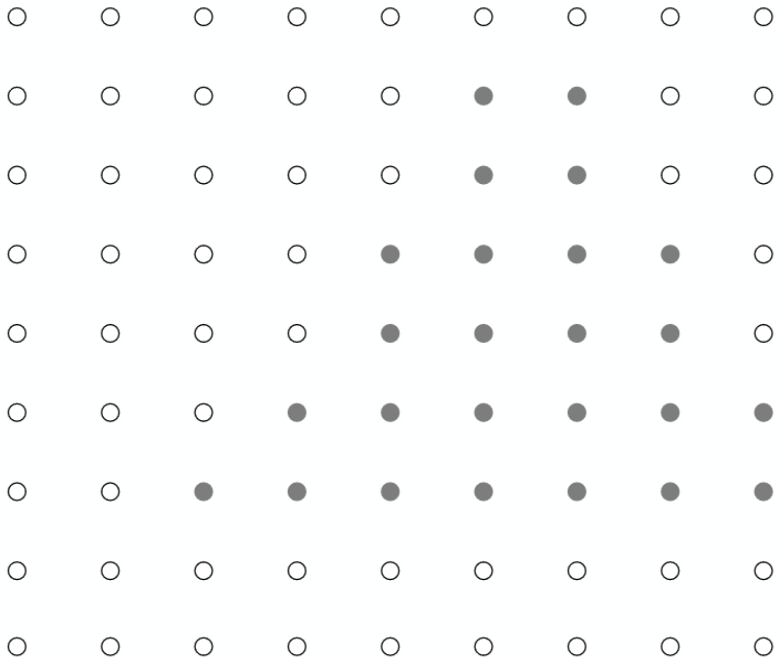
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

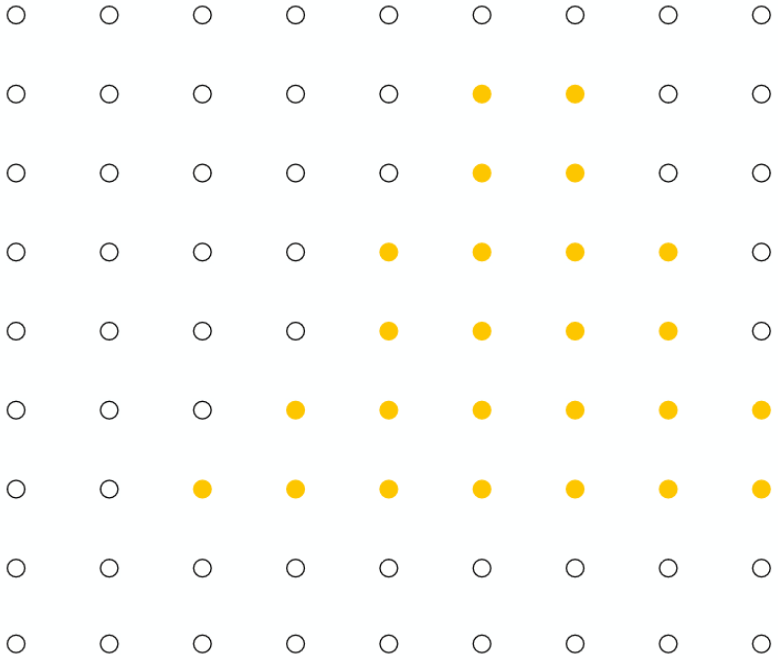


Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

Processing blue triangle:

depth = 0.75



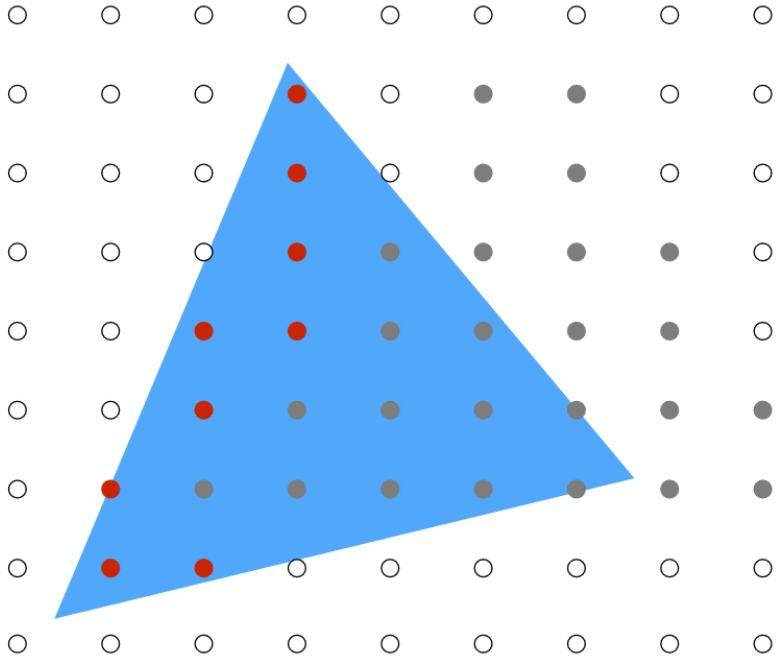
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

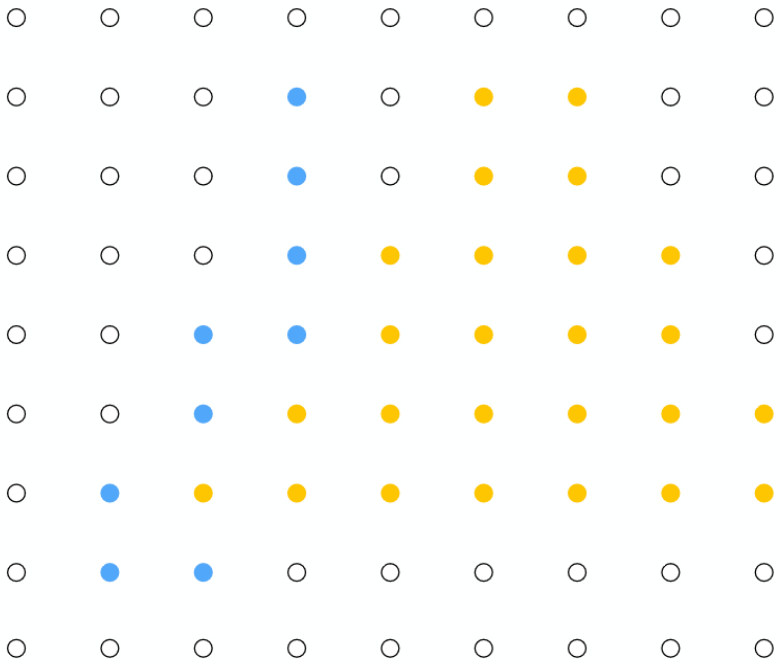
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing blue triangle:



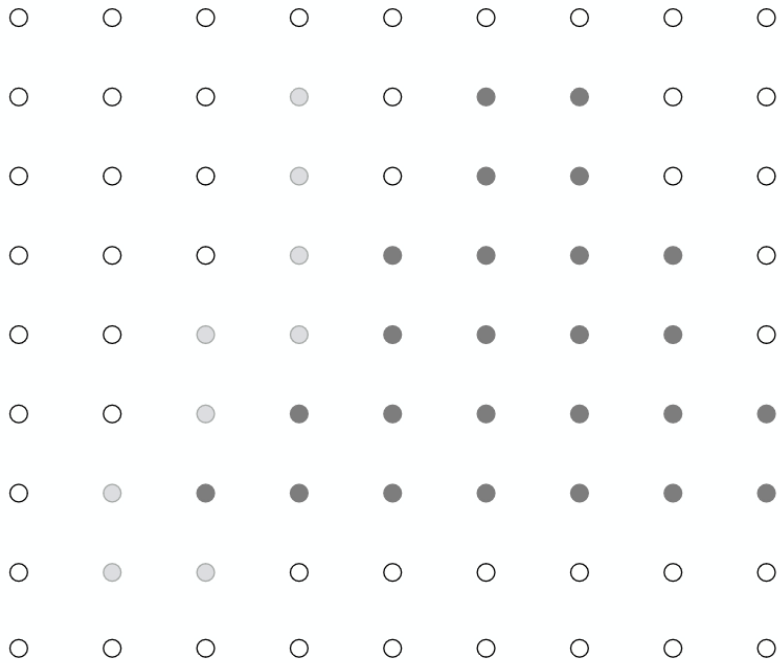
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

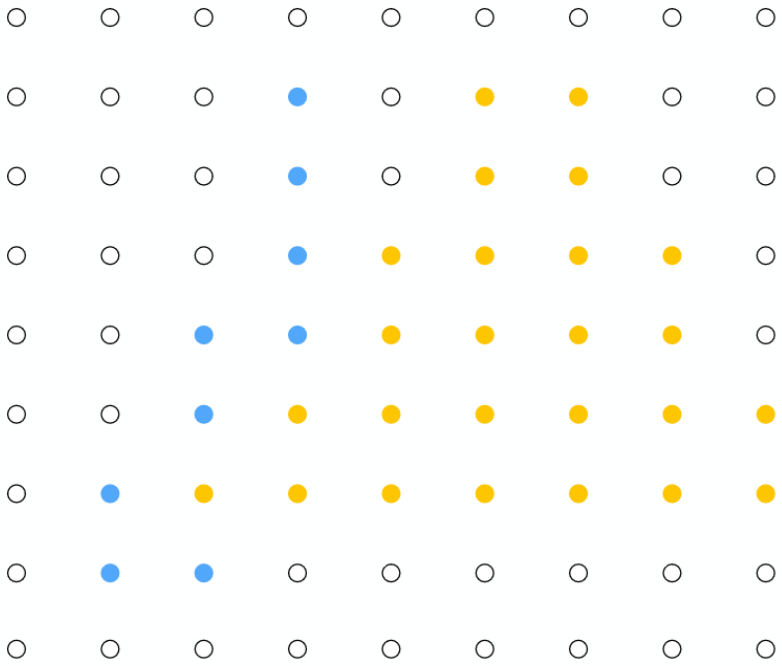


Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

Processing red triangle:

depth = 0.25



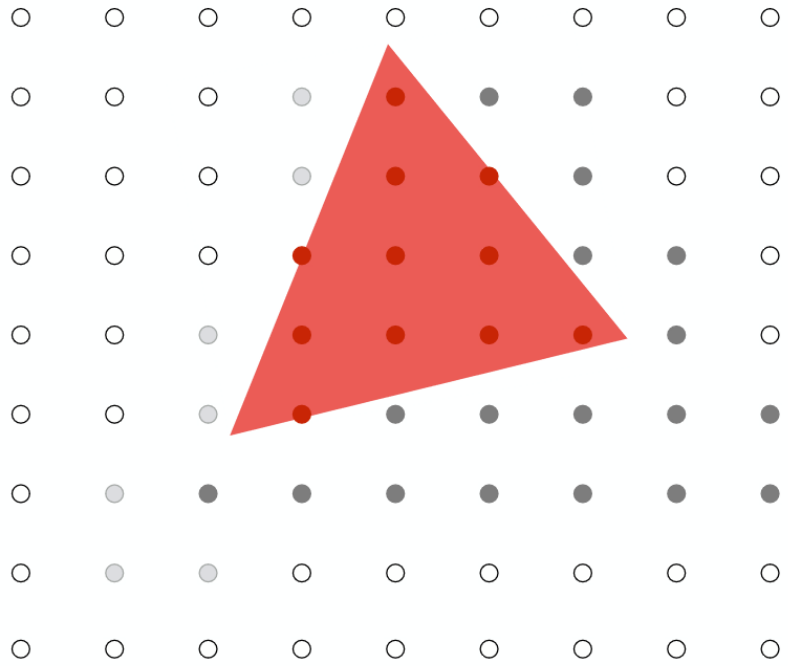
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

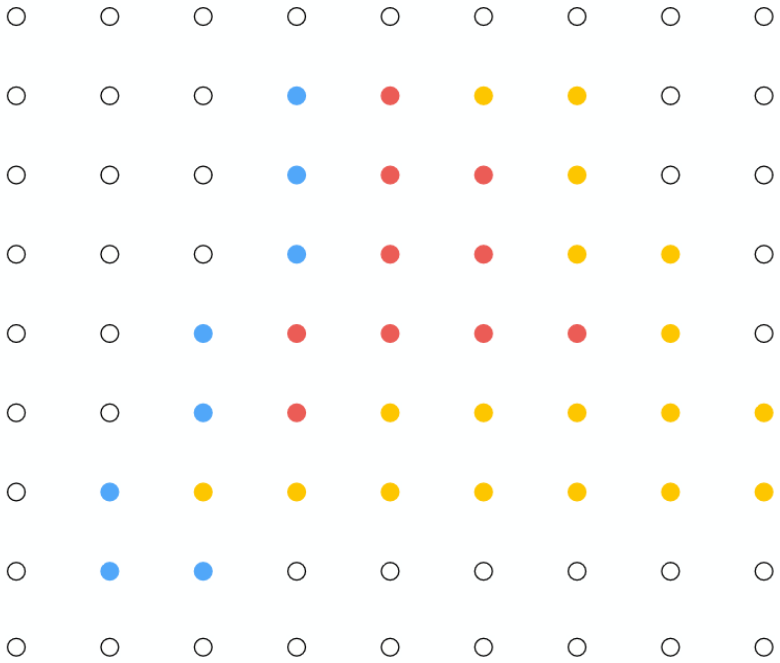
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing red triangle:



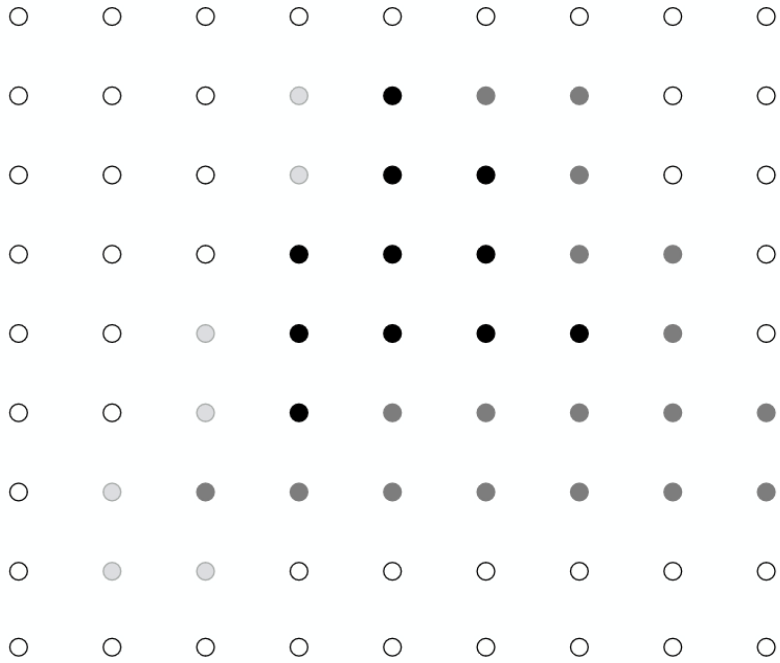
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

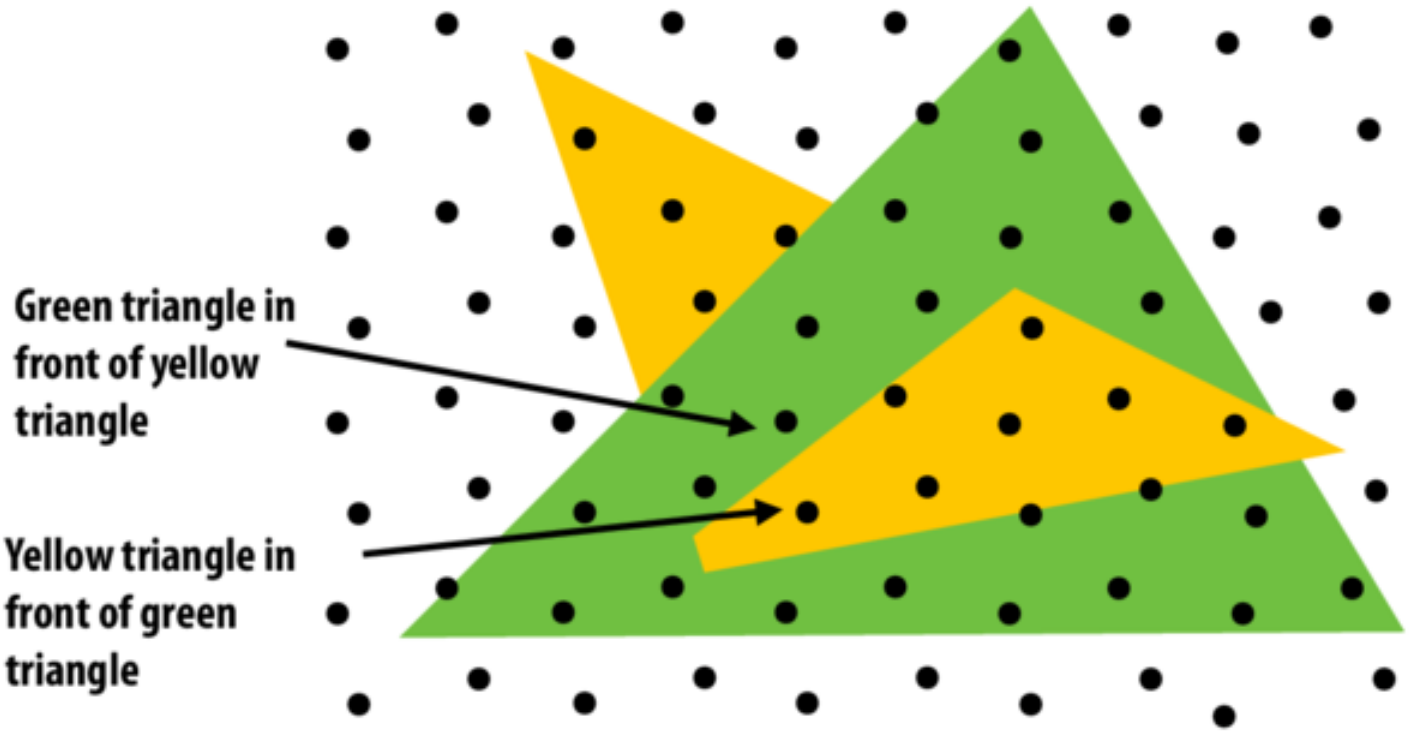


Depth buffer contents

Does depth-buffer algorithm handle interpenetrating surfaces?

Of course!

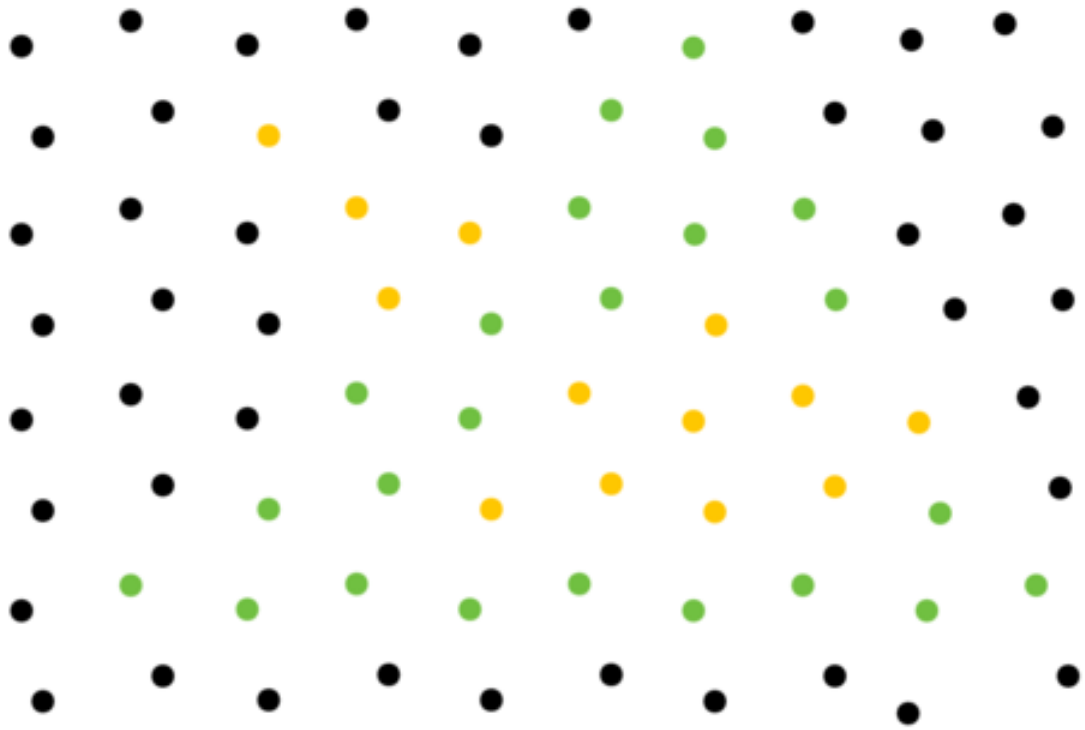
Occlusion test is based on depth of triangles at a given sample point. The relative depth of triangles may be different at different sample points.



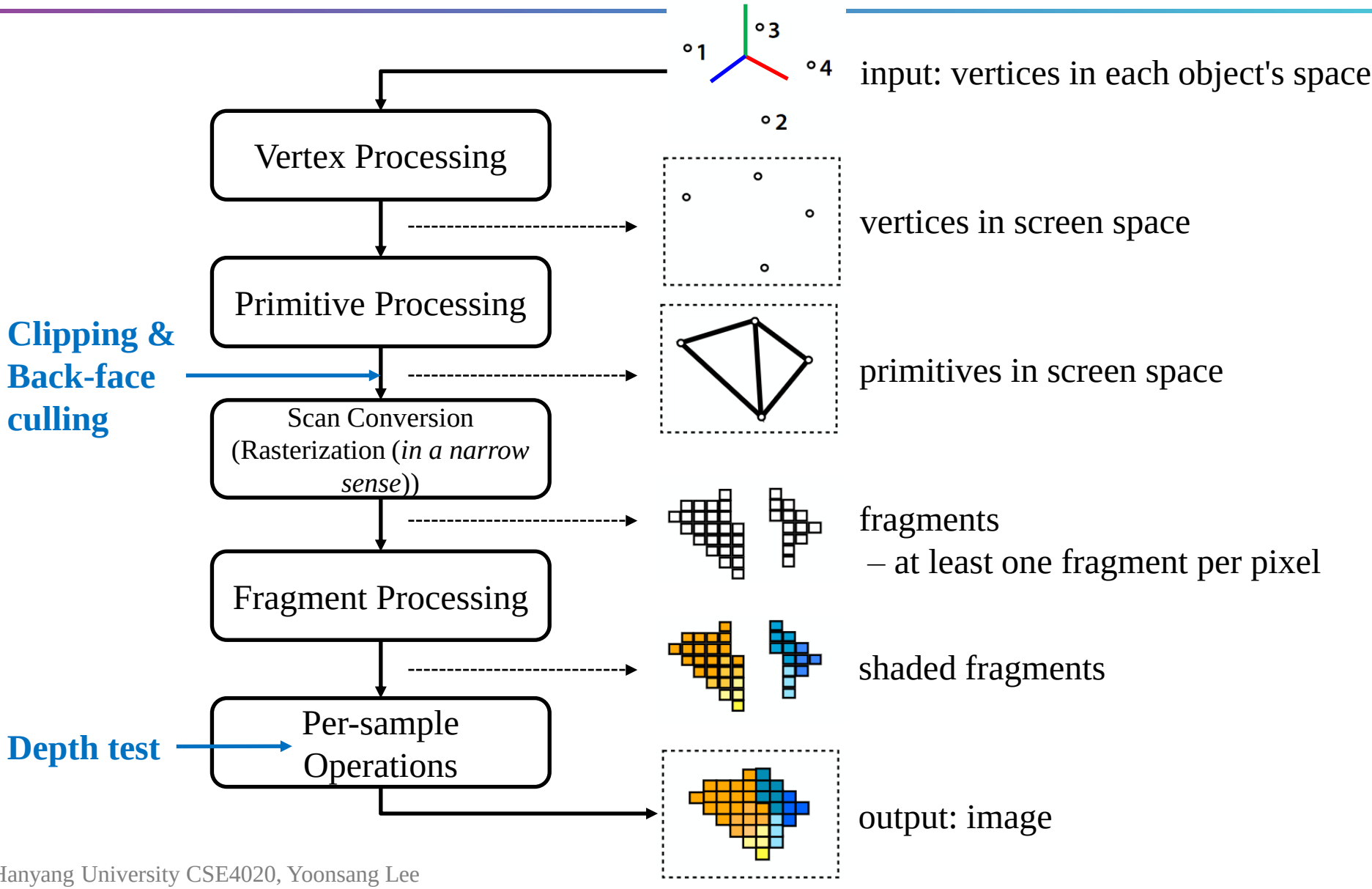
Does depth-buffer algorithm handle interpenetrating surfaces?

Of course!

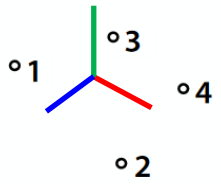
Occlusion test is based on depth of triangles at a given sample point. The relative depth of triangles may be different at different sample points.



Rendering Pipeline Again



Pipeline Input



input vertices in each object's space

Vertex Processing

transform matrices

Primitive Processing

textures, lighting model

Clipping & Back-face culling

Scan Conversion (Rasterization (*in a narrow sense*))

Fragment Processing

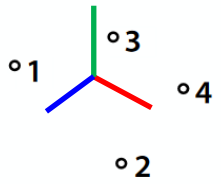
Depth test

Per-sample Operations

Pipeline inputs:

- **Input vertex data**
- **Programs that define vertex / fragment processing stages**: vertex / fragment shaders
- **Parameters for computing vertex positions in normalized coordinates**: transform matrices
- **Parameters for computing fragment color**: texture, lighting model

Pipeline Input



input vertices in each object's space

Vertex Processing

transform matrices

Primitive Processing

textures, lighting model

Clipping & Back-face culling

Scan Conversion (Rasterization (*in a narrow sense*))

Fragment Processing

Depth test

Per-sample Operations

- The black stages are automatically performed by the system.
- **The red stages are performed by the program you provide, using inputs you provide.**
- That is why we have been focusing on the red stages in this lecture, and the black stages were covered only for today.
- I believe you now have the ability to create your own software renderer that encompasses the entire process.

Lab Session

- We don't have a lab today!

Course Wrap-up

Do you remember?
- What is Computer Graphics?

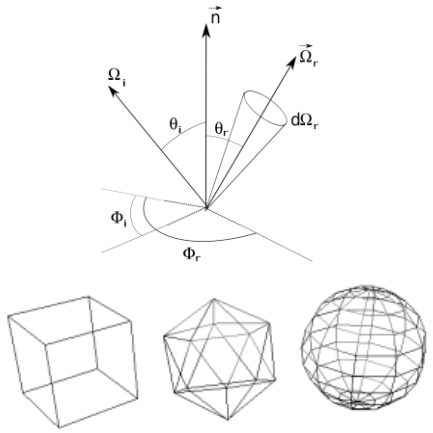
- The study of creating, manipulating, and displaying visual content using computers.

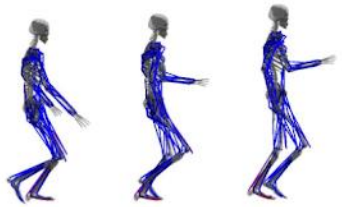
mathematical or geometrical models

Computer Graphics

Computer Vision

image or animation (a series of images)





Questions about Computer Graphics

- To do this, we should be able to answer:

- (Common:)
- How to express movement, placement, shape, and appearance of objects

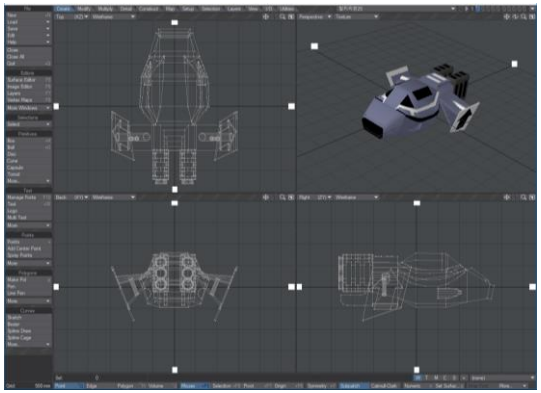
- (About rasterization:)
- How to map 3D objects into 2D screen
- How the whole rendering process is performed

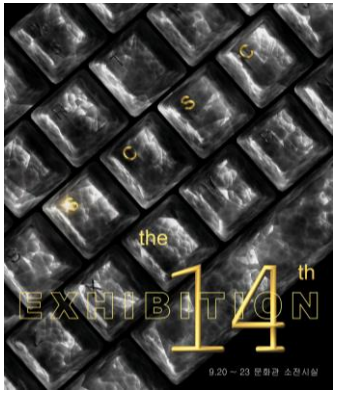
Movement & placement	3 - Transformations 4 - Affine Space Frame Matrix 7 - Hierarchical Modeling, Mesh 9 - Orientation & Rotation 10 - Character Animation 11 - Curves
Mapping to 2D screen	5 - Vertex Processing 1 6 - Vertex Processing 2
Shape	7 - Hierarchical Modeling, Mesh 11 - Curves
Appearance	8 - Lighting 12 - More Lighting, Texture
Rendering Pipeline	2 - Rendering Basics 13 - Scan Conversion, Visibility

How do you feel?

- If you've **had more fun** in this course than other courses, you already have **the potential** to do interesting research in computer graphics!

- I think, **passion** is the most important thing in computer graphics.
  - That was the starting point for me on this path.





If "computer graphics" truly feels enjoyable to you,

- + If you want to do a project related to character animation:
  - You can apply for a **graduation project** with me in your 4th year.

- + If you want to work more directly on character animation and study earlier:
  - Please email me: yoonsanglee@hanyang.ac.kr

Characteristics of Computer Graphics Research

- Requires a lot of **programming**.
  - If you like programming and have confidence, you are likely to do well.

- The fascinating aspect is that all the research findings are presented in a **visible** format.
  - In the field of computer graphics, a **video** is always attached when submitting a paper.

Computer Graphics and Robotics Lab.

- Website:
  https://cgrhyu.github.io/

- Youtube channel:
  https://www.youtube.com/@cgrlab

- Feel free to visit these sites anytime and take a look around.

Thanks for being a great class!