Computer Graphics

13 – Scan Conversion, Visibility

Yoonsang Lee
Hanyang University

Spring 2025

Final Exam Announcement (Unchanged)

- Date & time: **June 18 (Wed), 6:30–7:30 PM**
- Place: TBA
- Scope: **Lecture 8–12, Lab 8–12**

- **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!).

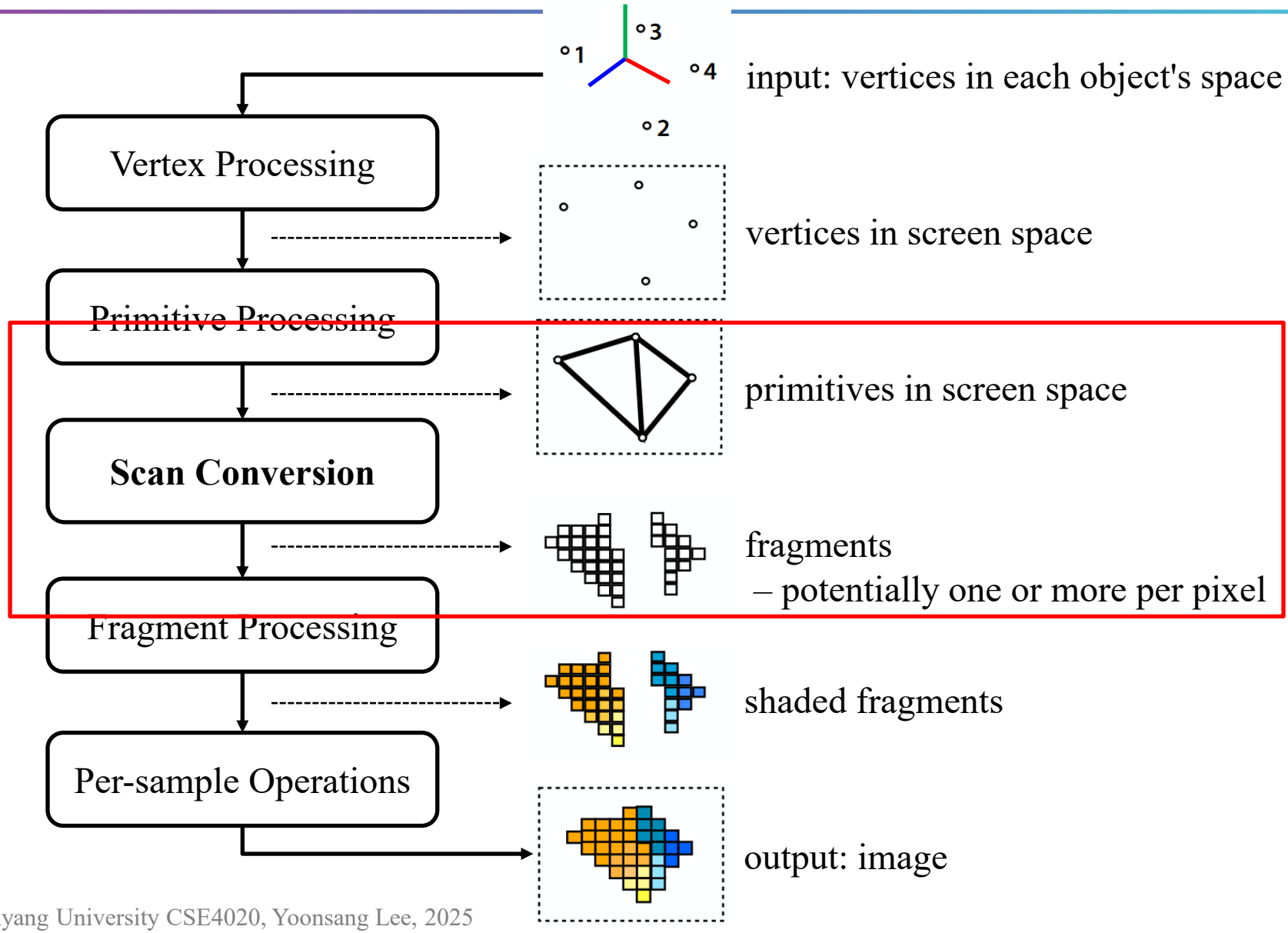
- Bring your **student ID card** to the exam.

Outline

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

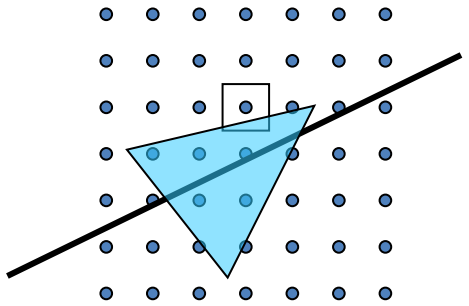
Scan Conversion

Recall: Rendering Pipeline



Scan Conversion

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



- First task: Determine which fragments are covered by a primitive.
  - A primitive refers to basic geometric shapes such as points, lines, circles, and polygons.

- Second task : Interpolate attribute values across the primitive.
  - e.g., color, normal, texture coordinates, etc.

Scan Conversion

- Algorithms that determine which fragments belong to a primitive are often referred to as 'drawing' algorithms.

- 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
  - Edge Function-Based Rasterization
  - Barycentric Coordinate Rasterization
  - Scanline Rasterization
  - Boundary fill
  - Flood fill
  - …
    - For details, refer to
    - https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage.html?utm_source=chatgpt.com
    - https://www.tutorialspoint.com/computer_graphics/polygon_filling_algorithm.htm

Scan Conversion

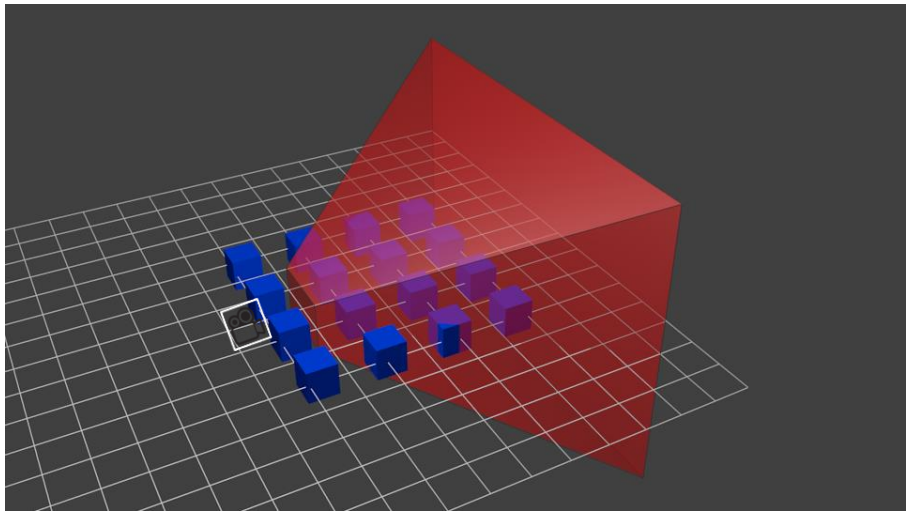
- We will not go into the details of these algorithms.
- In fact, these tasks are more complex than they may initially seem.
  - Computational efficiency, anti-aliasing, ...
- However, most graphics APIs (such as OpenGL) provide built-in support for these operations, which are now efficiently handled by modern graphics hardware (GPUs).
  - These algorithms were extensively studied in the early days of computer graphics, and are now quite mature.
- I believe today you can simply “draw” primitives using graphics APIs without worrying about the underlying details.

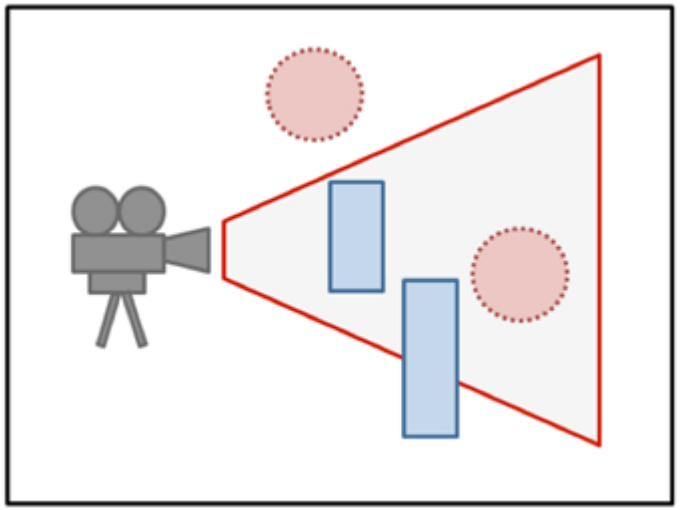
Visibility Problem

Visibility Problem

- What is VISIBLE?

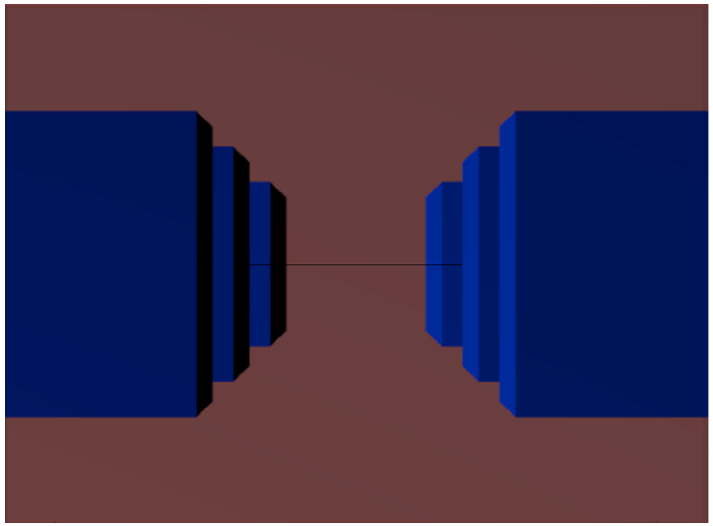
Red: viewing frustum, Blue: objects

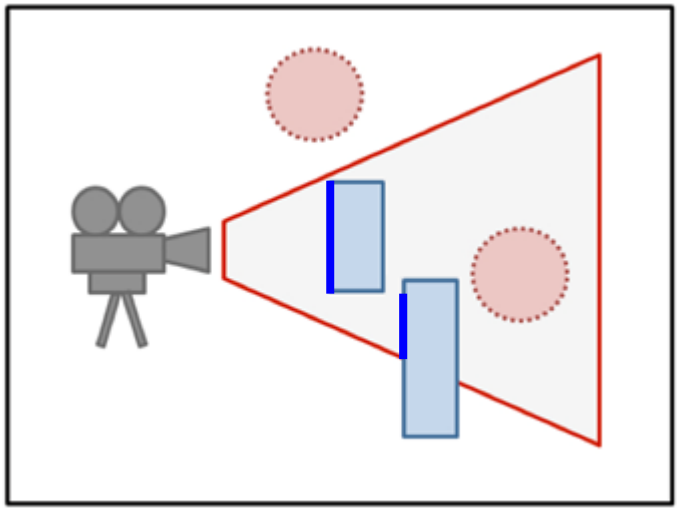




Visibility Problem

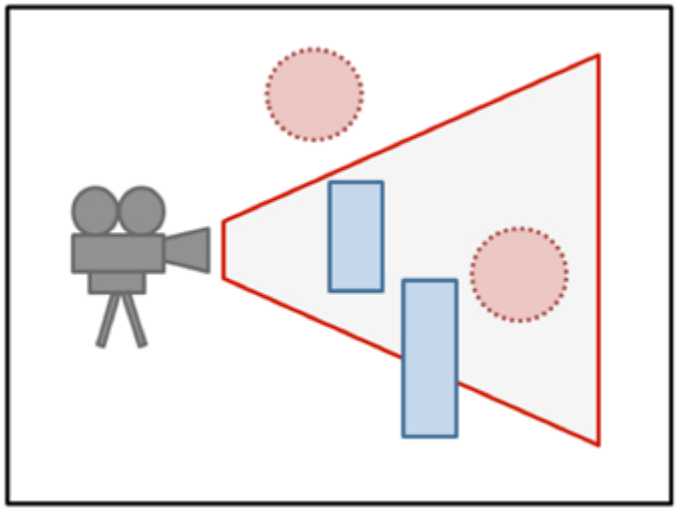
- The answer is:





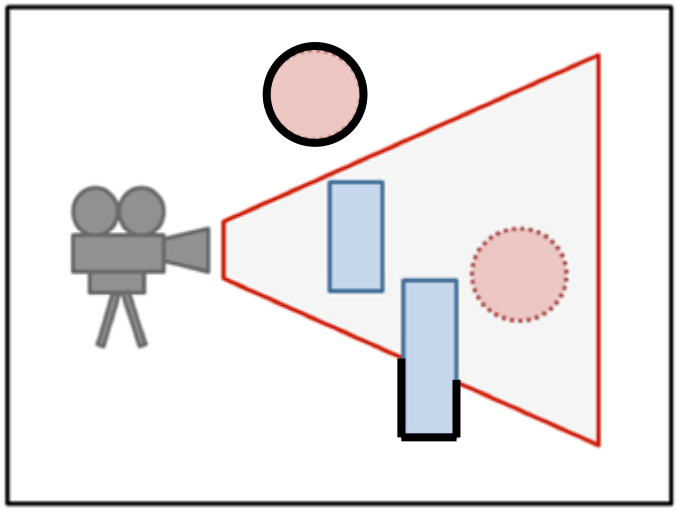
Visibility Problem

- What is NOT VISIBLE?



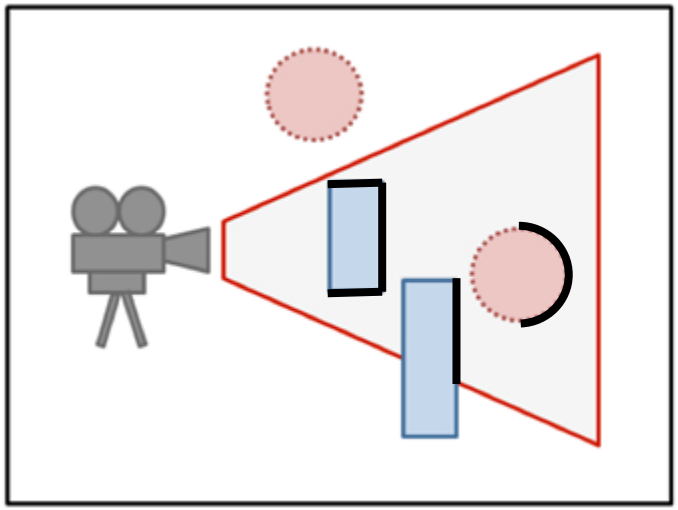
Visibility Problem

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



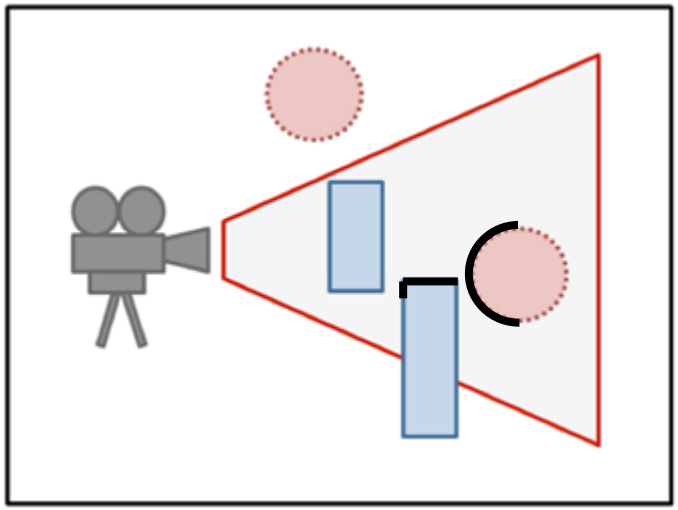
Visibility Problem

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



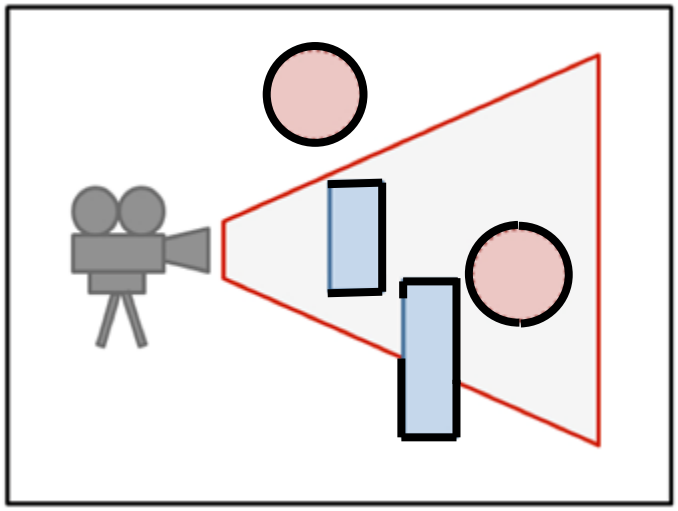
Visibility Problem

- 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

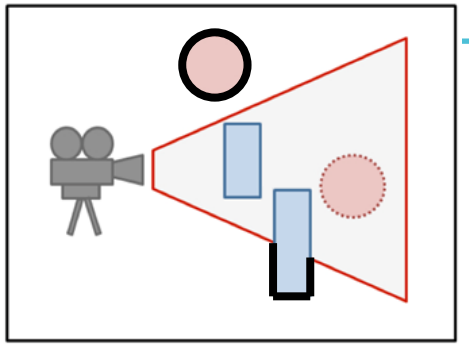
- Removing...

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

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

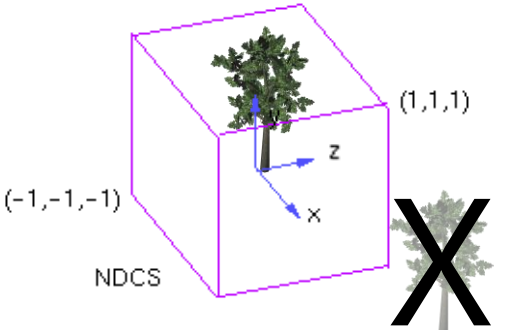
- Primitives occluded by other objects closer to the camera
- → **Hidden surface removal**

Clipping (Viewing Frustum Culling)

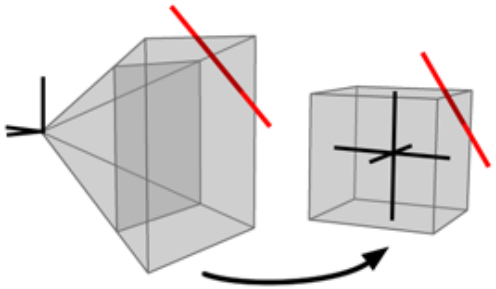


- Removing primitives outside the viewing frustum

- 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 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 is called "clip space".

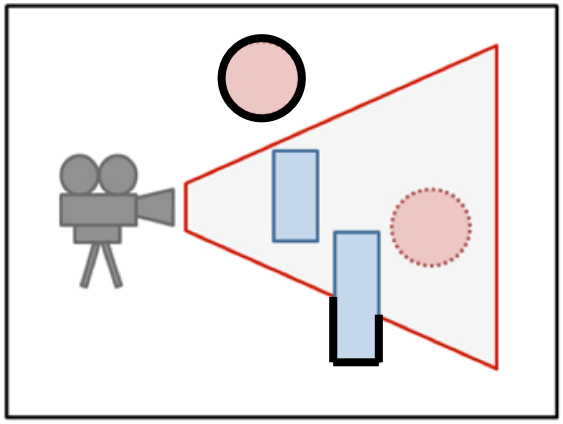


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix} \sim \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$



Clipping (Viewing Frustum Culling)

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

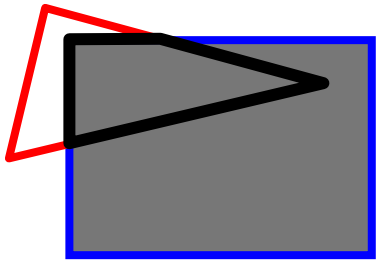


- Polygon clipping algorithms
  - Sutherland–Hodgman (1974)
  - Weiler–Atherton (1977)
  - ...

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

Clipping (Viewing Frustum Culling)

- These tasks, like drawing algorithms, are also more complex than they may initially appear — especially polygon clipping.
  - Vertices may be added or removed when clipping a triangle.



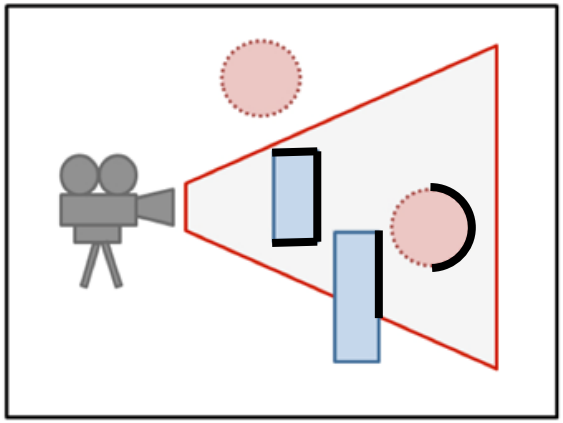
triangle → quad

- Once again, we'll omit the details of these algorithms.

- Clipping is automatically handled by the GPU once the view frustum is set through the graphics API (such as OpenGL).

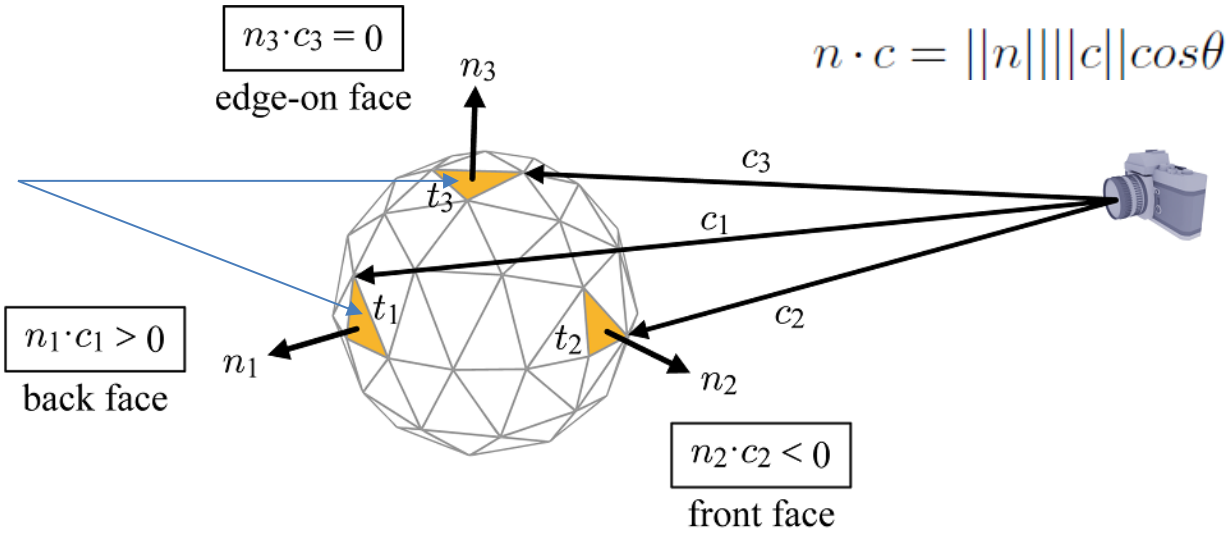
Back-Face Culling

- Removing back-facing primitives



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

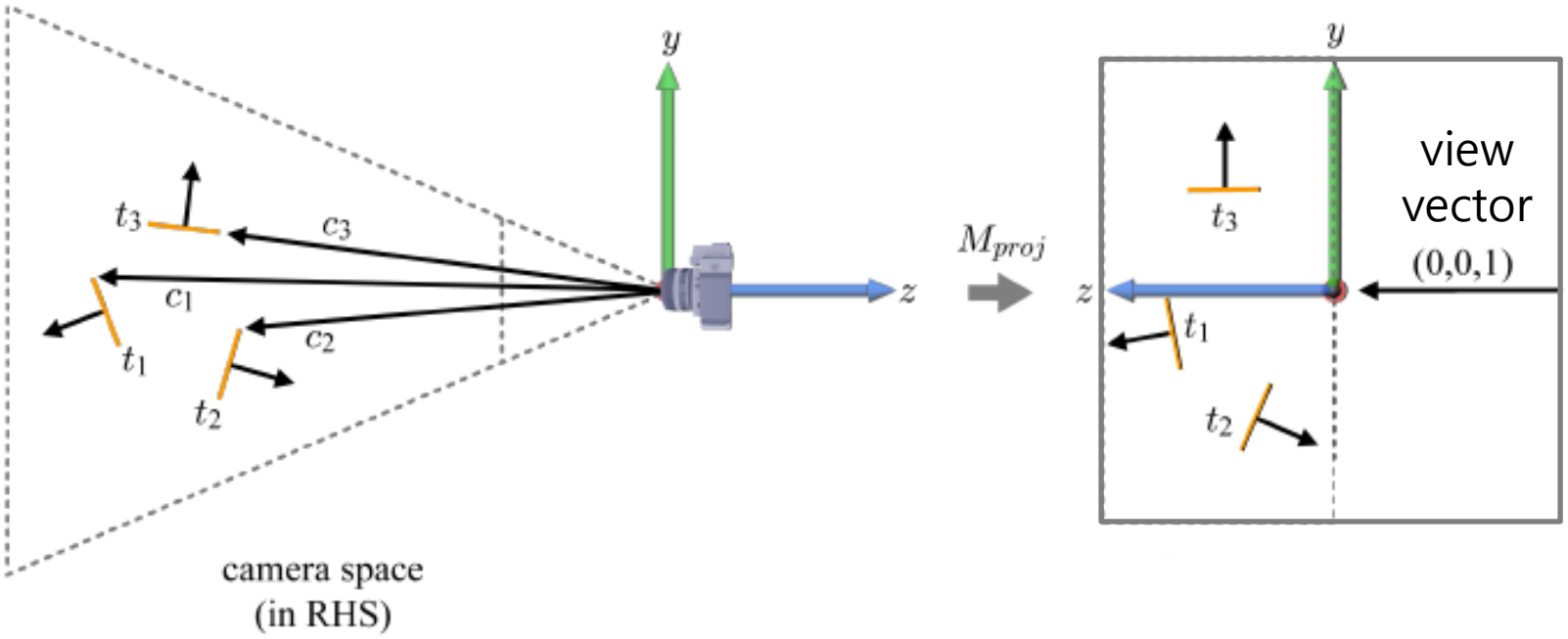
Discarded!



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

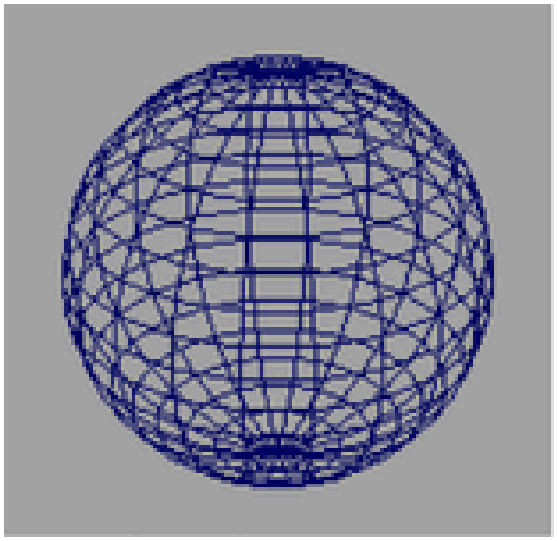
Back-Face Culling

- 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.

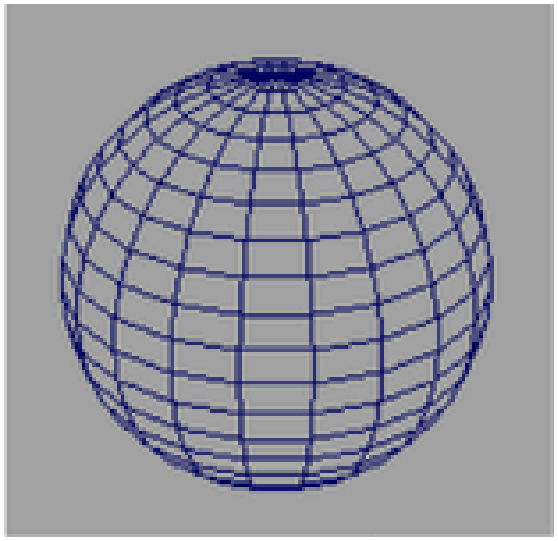


* This image is from the slides of Prof. JungHyun Han (Korea Univ.): http://media.korea.ac.kr/book/

Back-Face Culling



Backfaces

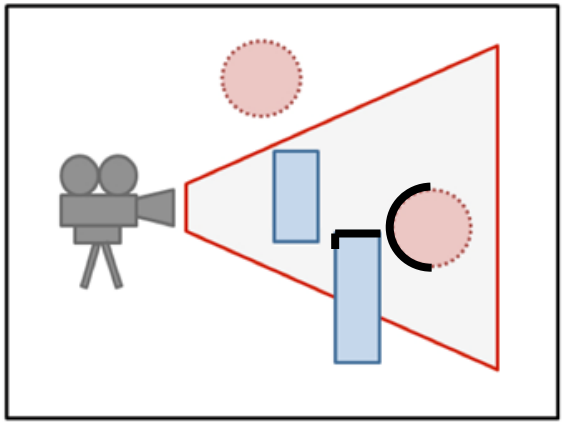


No backfaces

* 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

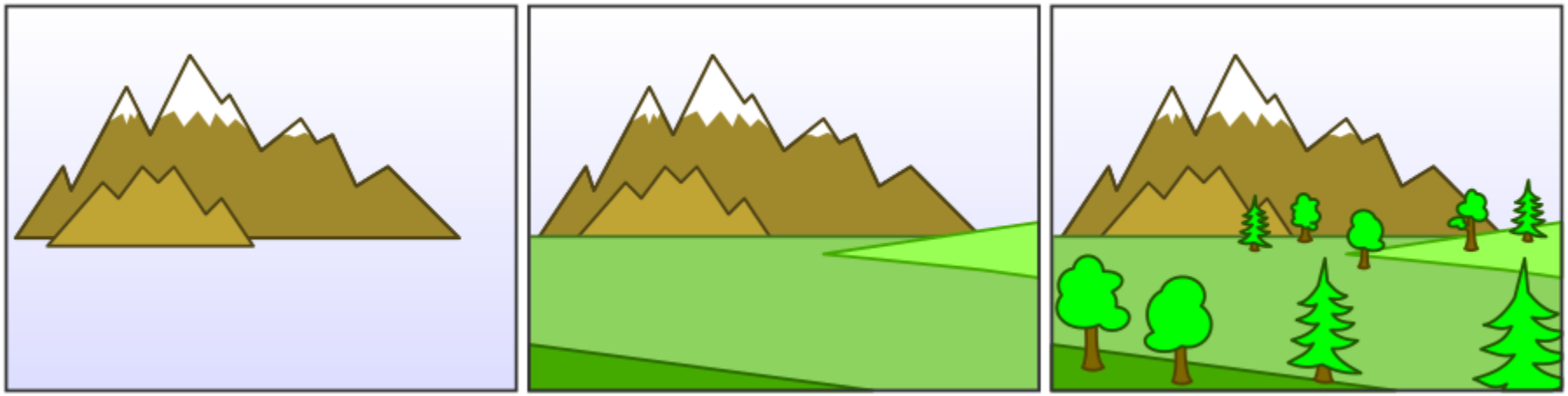
- There are several algorithms for hidden surface removal, including:
  - Z-buffering (a.k.a. Depth buffering)
  - Painter's algorithm
  - BSP tree traversal
  - ...

- Z-buffering has become the standard technique in modern graphics systems.

- Now, let's explore the basic concepts behind the Painter's algorithm and Z-buffering.

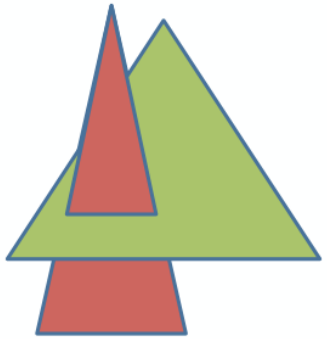
Painter's Algorithm

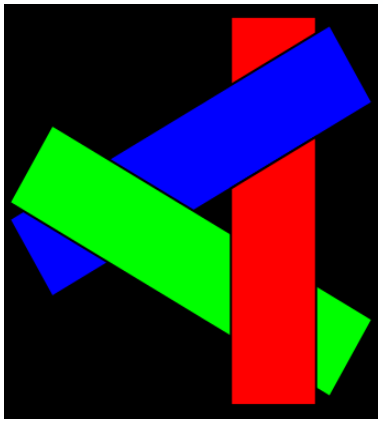
- Sorts all polygons based on their distance from the viewer, or *depth*, and then paints them in this order, farthest to closest.
- Polygons that are closer to the viewer will be drawn on top of polygons that are farther away.
- Works on a polygon-by-polygon basis.



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

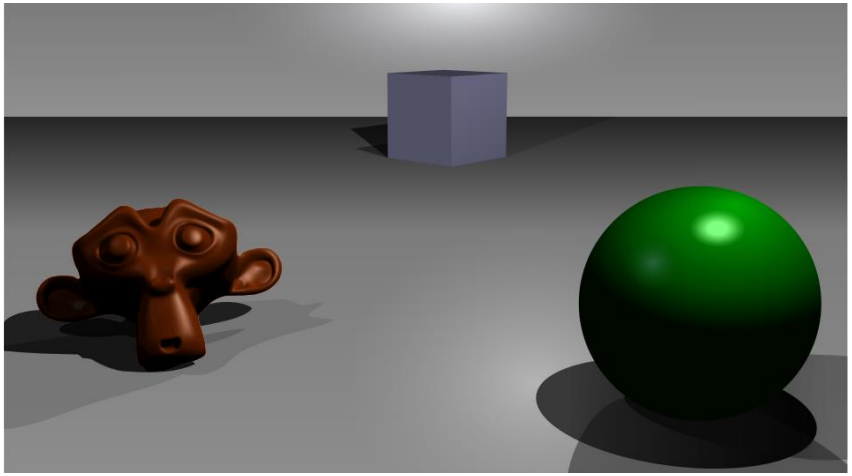
- In this method, a separate buffer called the ***z-buffer*** (a.k.a. ***depth buffer***) is used to store the depth value of each pixel (fragment) on the screen.
  - Z-buffering is often described as working per pixel, but technically, depth comparison happens per fragment.
- During rendering, **for each pixel**, the z-buffer is checked to determine whether the newly rendered object is closer to the viewer than the one already stored.
  - If it is closer, the new object is drawn at that pixel, and corresponding depth value is updated in the z-buffer.
  - If it is farther away, the new object is discarded and the existing object's color at that pixel remains unchanged.
- Works on a pixel-by-pixel basis.

Z-Buffering: Algorithm

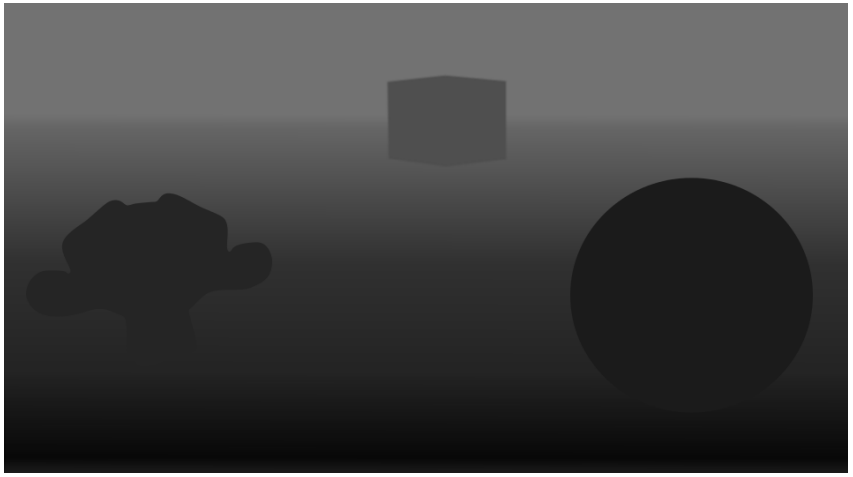
```
allocate depth_buffer;          // Allocate depth buffer → Same size as viewport.

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.
```

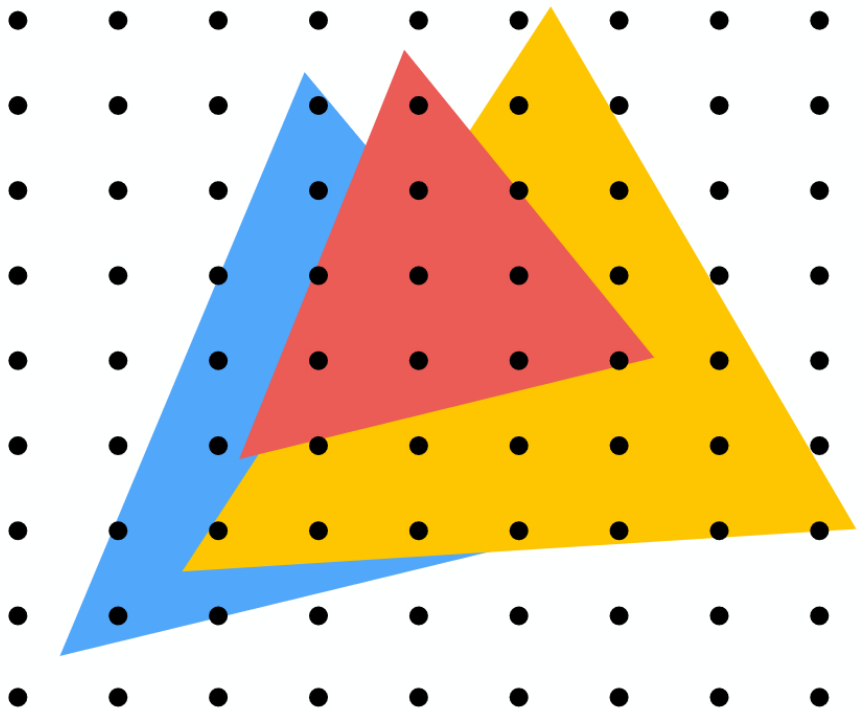


Frame buffer



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

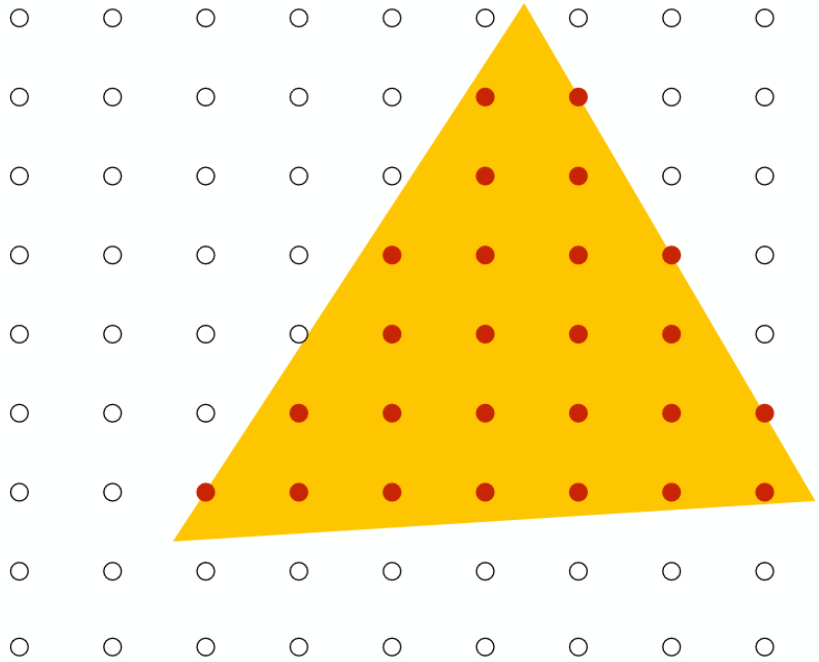
White = large distance

Black = small distance

Red = sample passed depth test

Color buffer contents

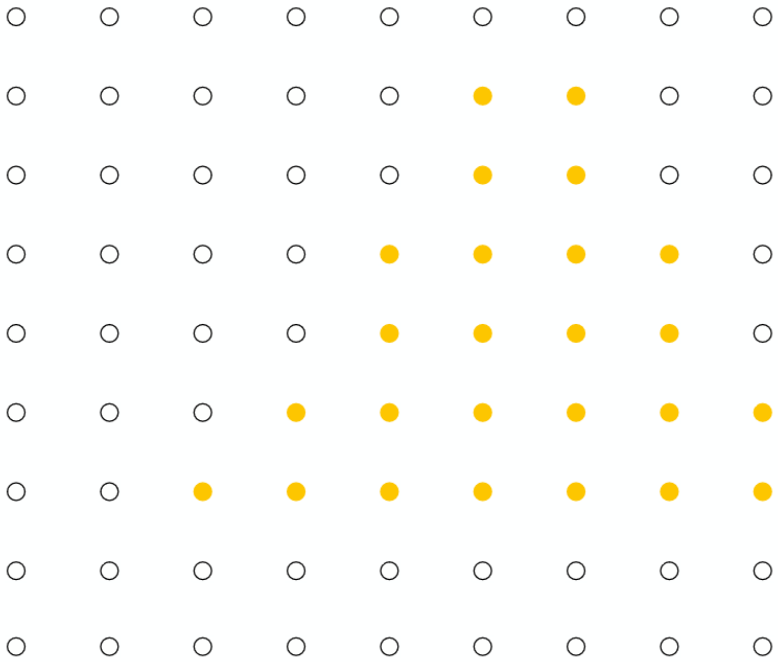
= Frame buffer



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing yellow triangle:



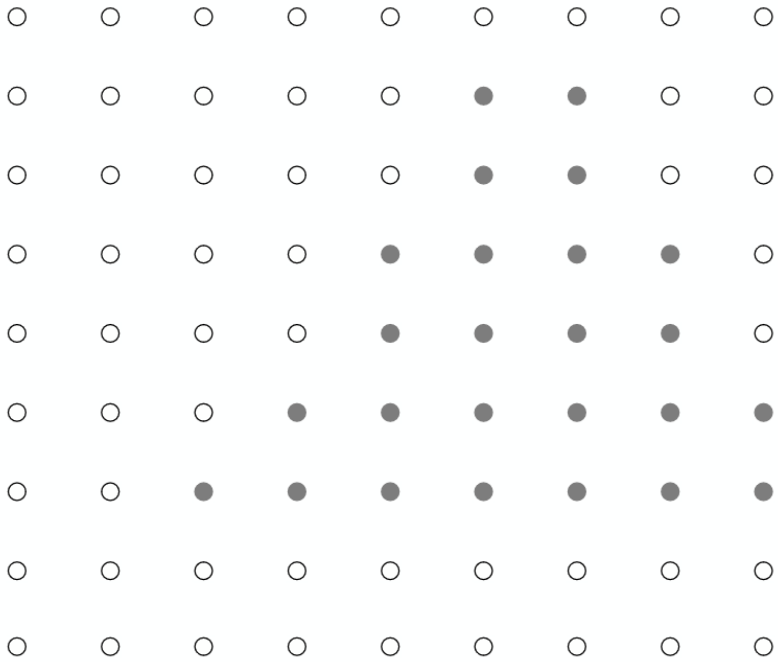
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

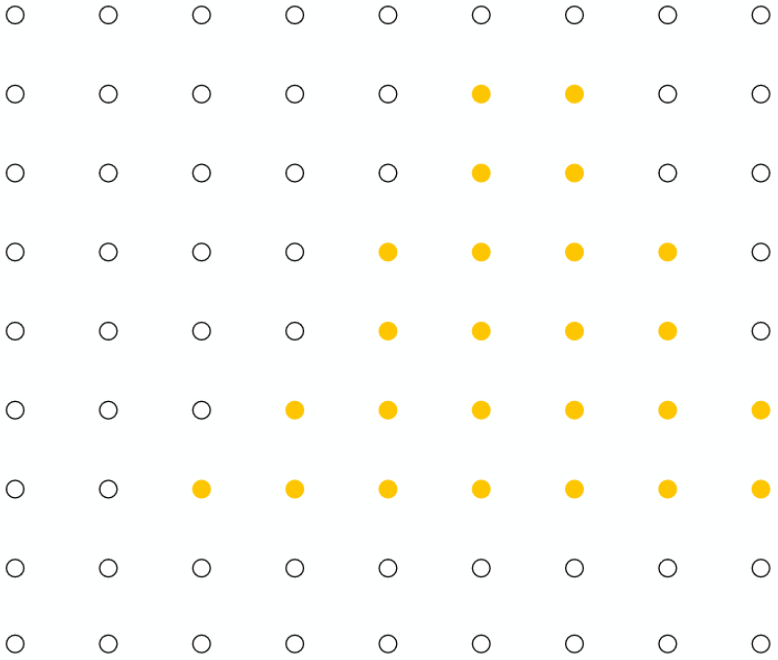


Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

Processing blue triangle:

depth = 0.75



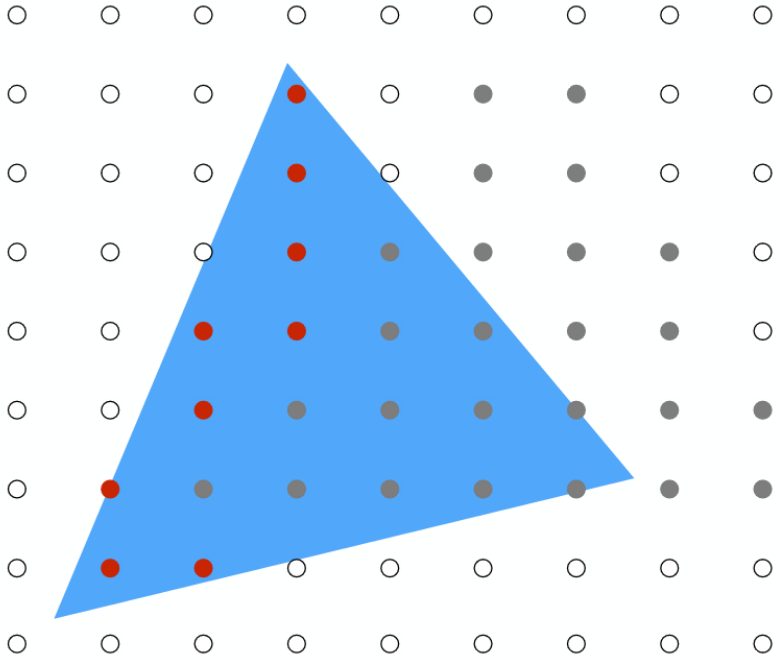
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

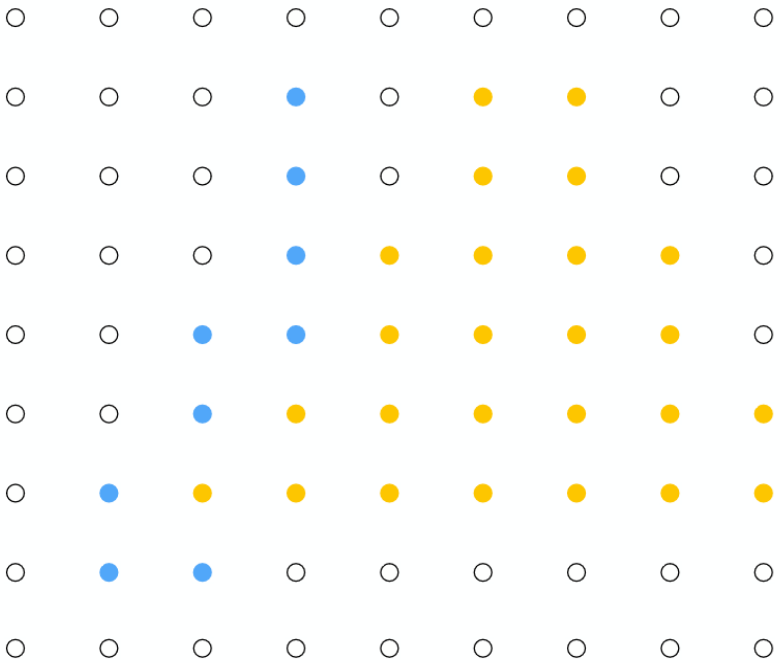
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing blue triangle:



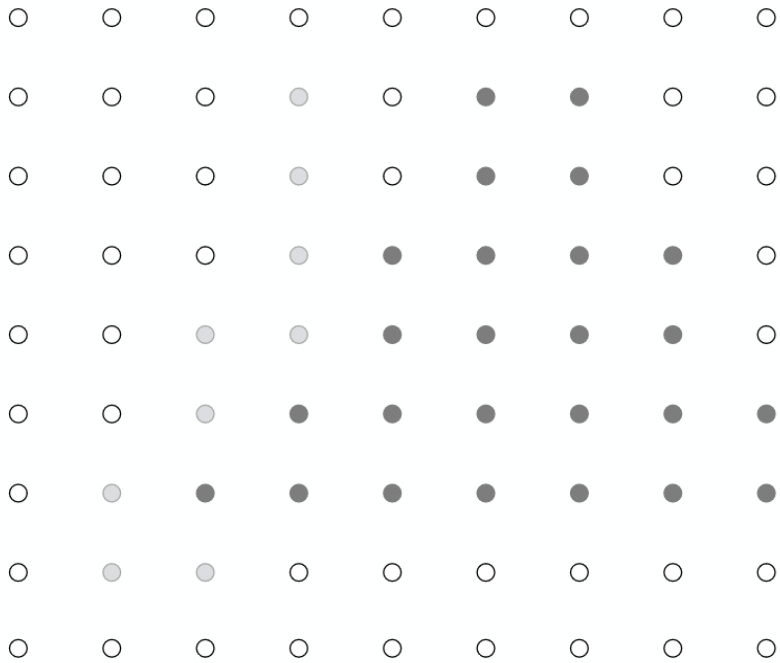
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

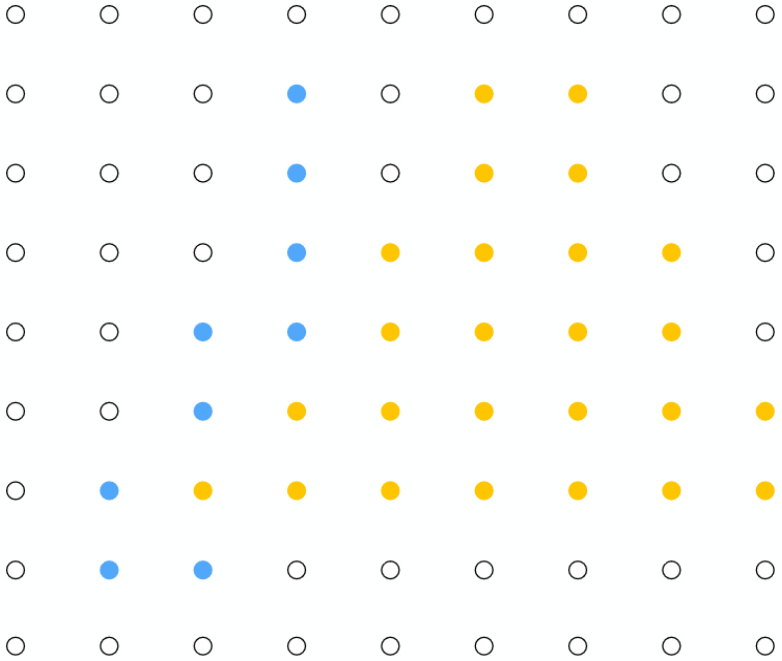


Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

Processing red triangle:

depth = 0.25



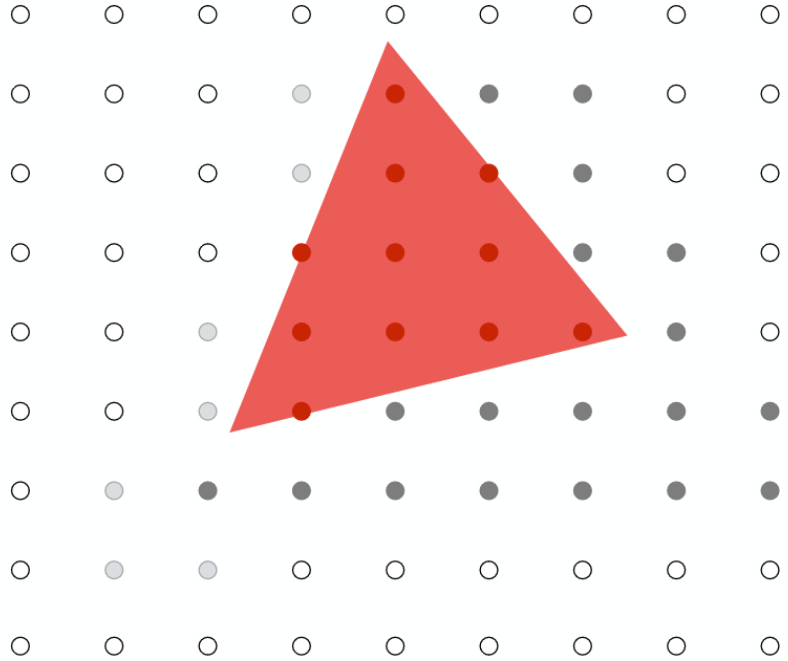
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

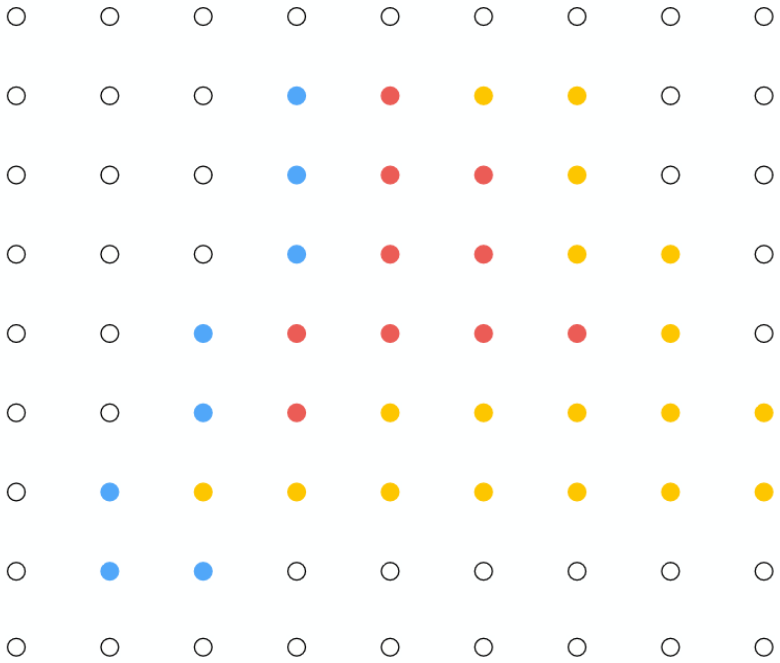
Red = sample passed depth test



Depth buffer contents

Occlusion using the depth-buffer (Z-buffer)

After processing red triangle:



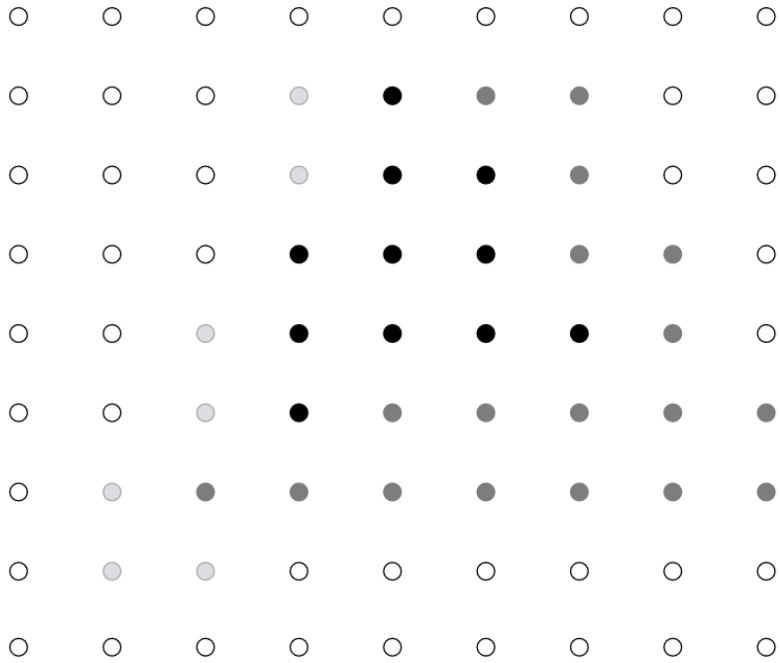
Color buffer contents

Grayscale value of sample point used to indicate distance

White = large distance

Black = small distance

Red = sample passed depth test

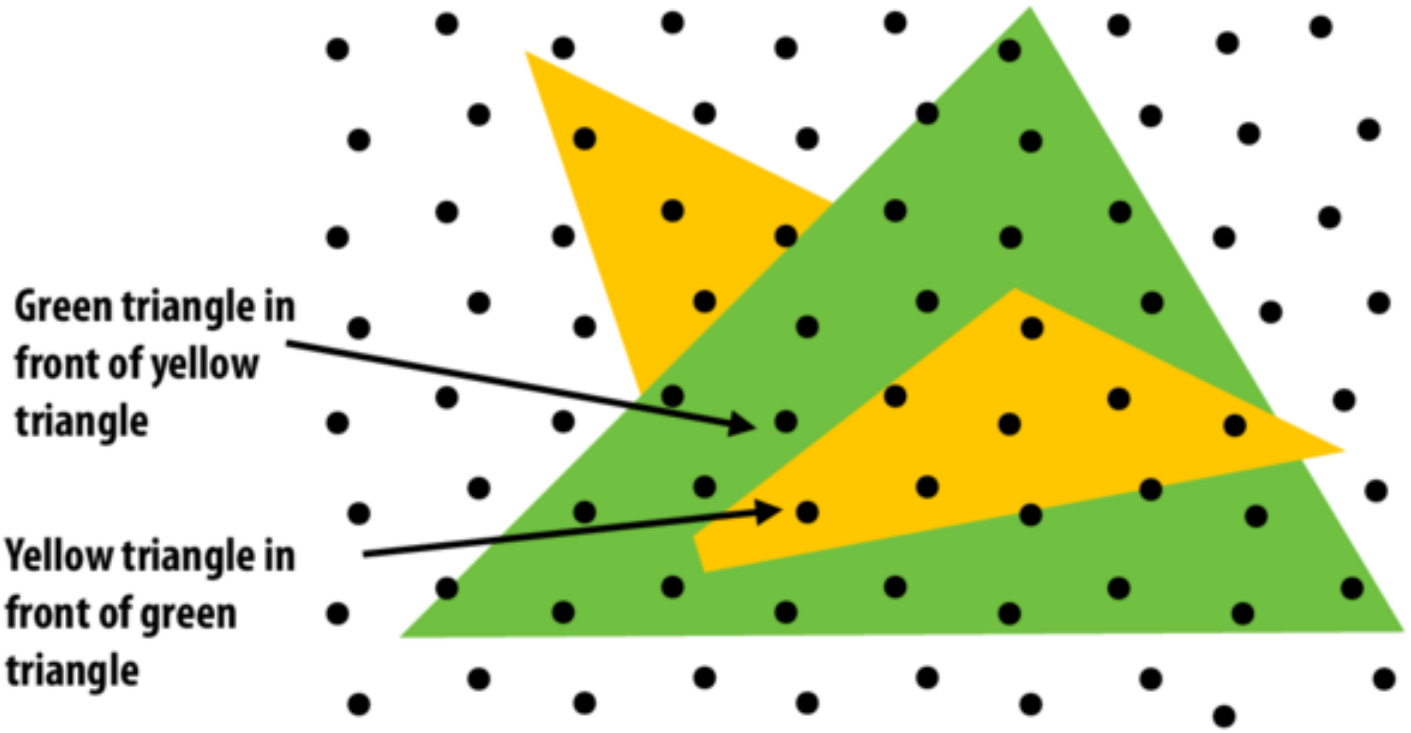


Depth buffer contents

Does depth-buffer algorithm handle interpenetrating surfaces?

Of course!

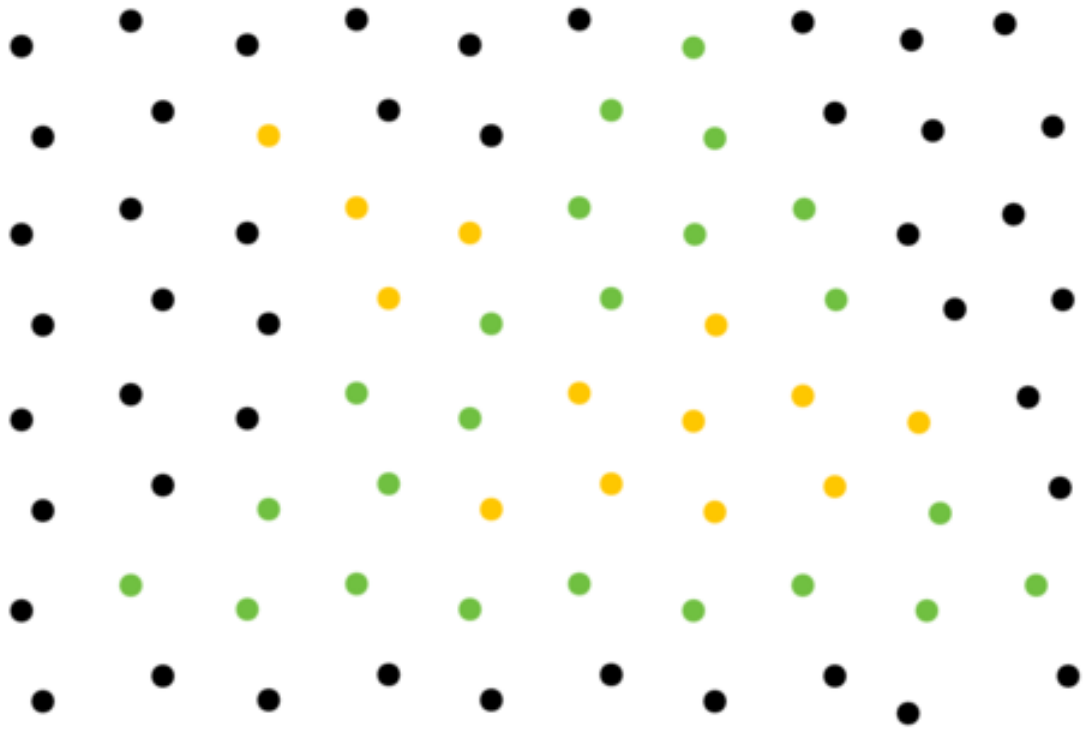
Occlusion test is based on depth of triangles at a given sample point. The relative depth of triangles may be different at different sample points.



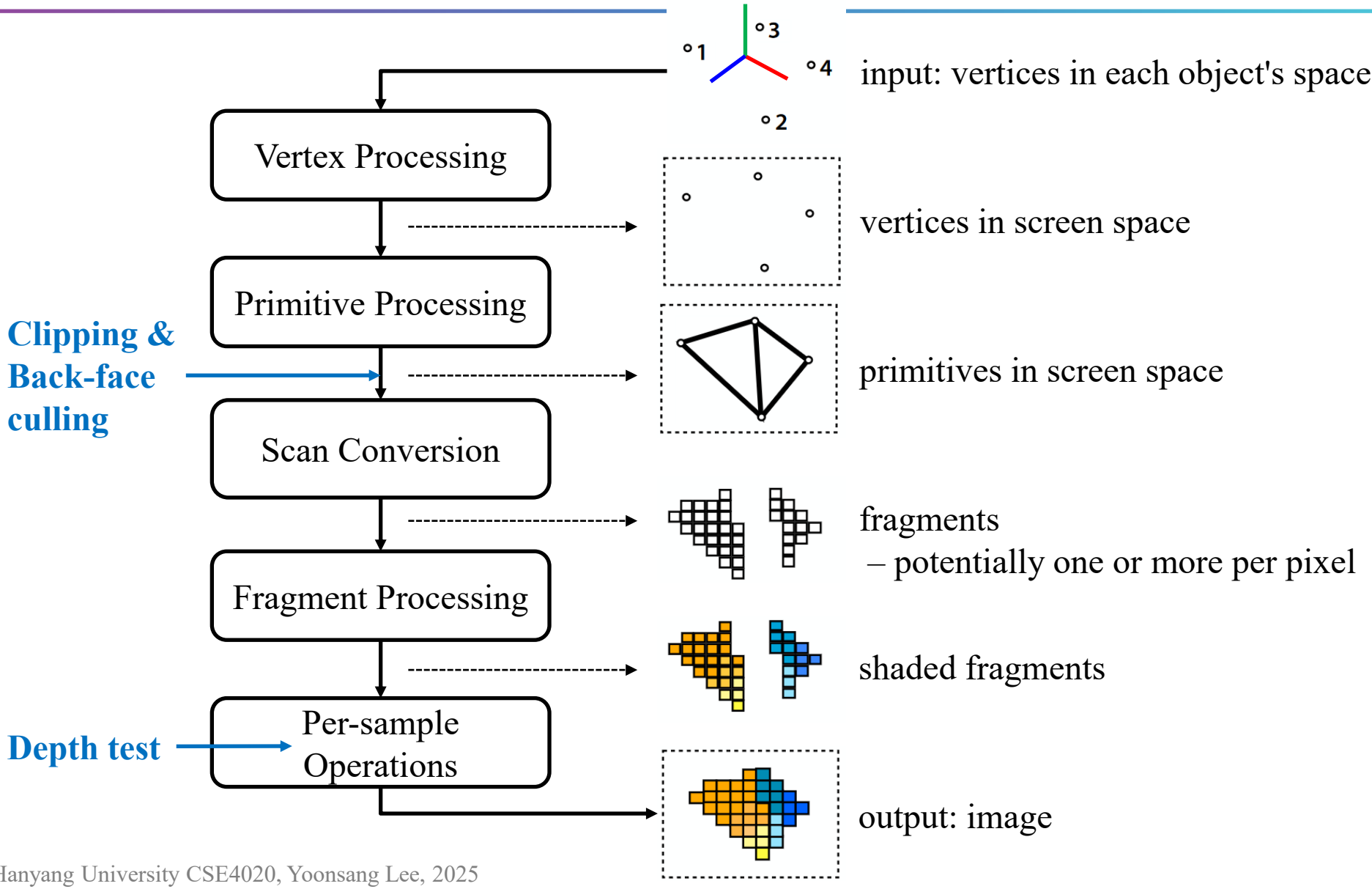
Does depth-buffer algorithm handle interpenetrating surfaces?

Of course!

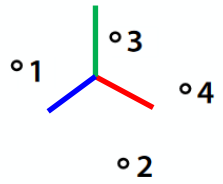
Occlusion test is based on depth of triangles at a given sample point. The relative depth of triangles may be different at different sample points.



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

Fragment Processing

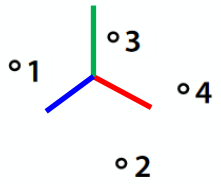
Depth test

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

Clipping & Back-face culling

Scan Conversion

textures, lighting model

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 throughout the semester, and only covered the black ones 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?

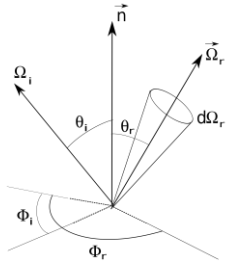
- The study of creating, manipulating, and displaying visual content using computers.

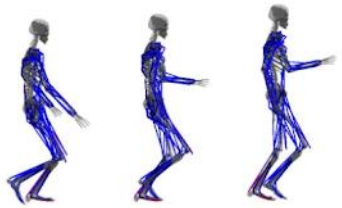
mathematical or geometrical models

Computer Graphics →

← Computer Vision

image or animation (a series of images)





Questions about Computer Graphics

- To achieve this, we need to be able to answer the following questions:

- (General concepts)
- How can we represent the movement, placement, shape, and appearance of objects?

- (Rasterization-specific)
- How do we project 3D objects onto a 2D screen?
- How is the entire rendering process carried out?

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 Was It for You?

- It may have been tough,
- but I hope you had **fun** and felt a **sense of reward**—especially during the assignments and the project.

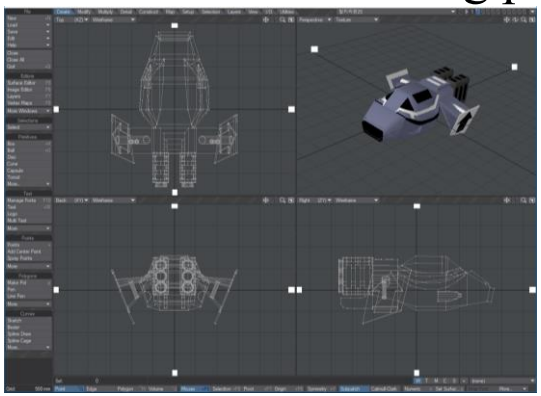
- I also understand that some of you might feel discouraged (I know, I know.)
  - maybe because of low exam scores,
  - or because your project was not successful.

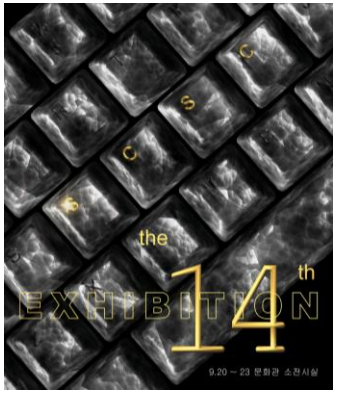
It’s Okay to Feel That Way

- What really matters is:

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

- In computer graphics, **passion** is what drives creativity and progress — it's more important than anything else.
  - That was the starting point for me on this path.





Take Time to Discover What Feels Right for You

- I truly hope that each of you takes time to reflect—
  - on what you enjoy, what you're curious about, and what kind of work brings you energy.

- If this course — or perhaps another one — can support you even a little along that journey, then it will have been more than worthwhile.

If You Enjoyed This Course, What's Next?

- If you're interested in projects related to character animation or robotics,
  - You can apply for a **graduation project** with me in your 4th year.

- If you'd like to get involved earlier and explore those fields,
  - Feel free to email me at: yoonsanglee@hanyang.ac.kr

Characteristics of Computer Graphics Research

- Requires a lot of programming,
  but don’t worry—you don’t have to be an expert to get started.
  - If you enjoy solving problems and are willing to learn, you'll do just fine.

- One of the most fascinating aspects is that research in this field is **highly visual.**
  - In computer graphics, papers are almost always accompanied by a **video**, which makes your work easy to share and exciting to watch.

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.

One Last Thing…

- I hope this course has been meaningful and helpful

  — not only in learning computer graphics,

  but also in finding what drives you, what matters to you, and maybe even becoming a small turning point in your twenties.

Thanks for being a great class!