CSE 167:
Introduction to Computer Graphics
Lecture #8: Visibility Culling

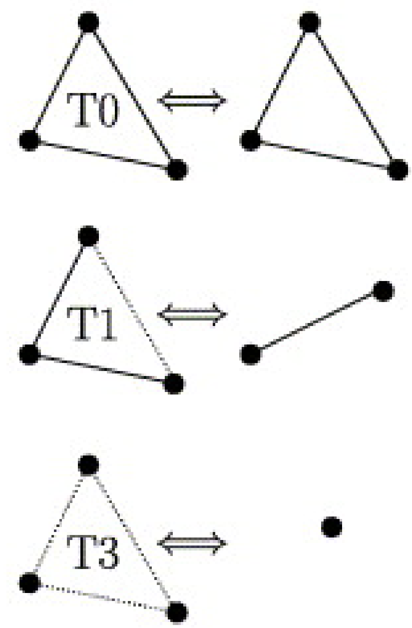
Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2019

Small Object Culling

- Object projects to less than a specified size
  - Cull objects whose screen-space bounding box is less than a threshold number of pixels

Degenerate Culling

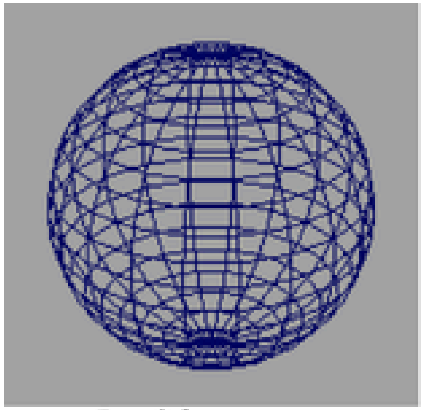
- Degenerate triangle has no area
  - Normal **n**=0
  - All vertices in a straight line
  - All vertices in the same place

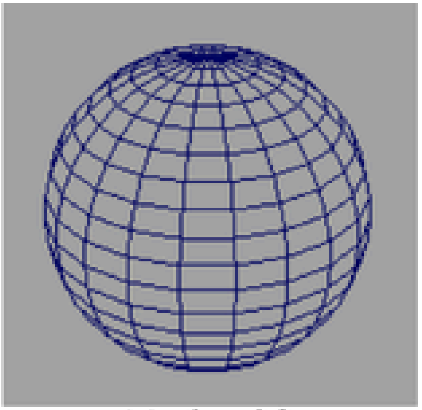


Source: Computer Methods in Applied Mechanics and Engineering, Volume 194, Issues 48–49

Backface Culling

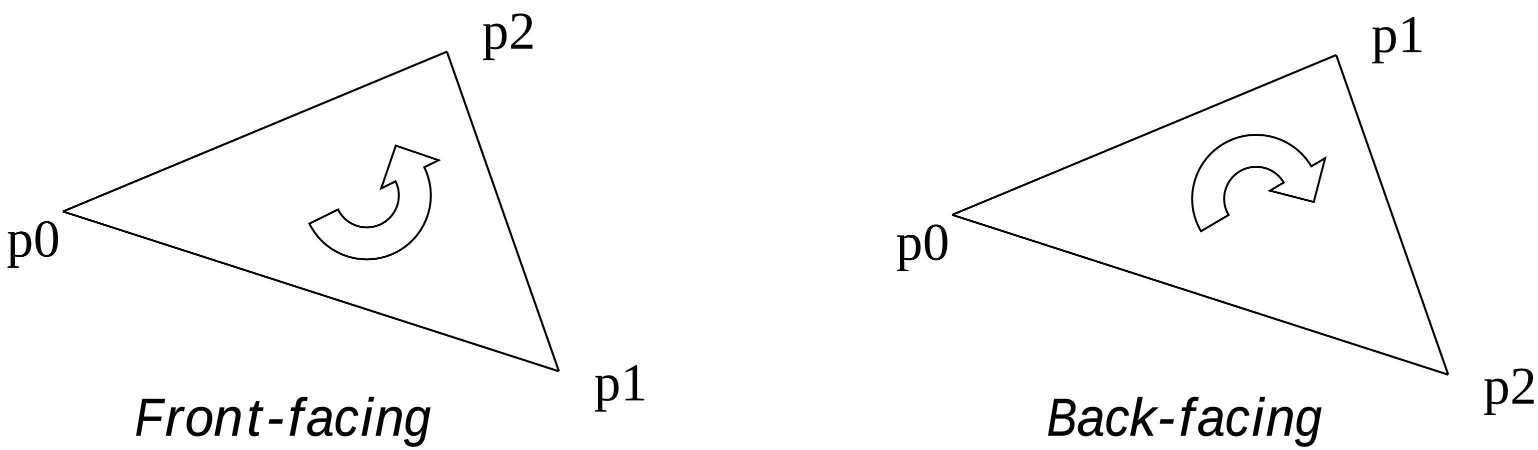
- Consider triangles as "one-sided", i.e., only visible from the "front"
- Closed objects
  - If the "back" of the triangle is facing away from the camera, it is not visible
  - Gain efficiency by not drawing it (culling)
  - Roughly 50% of triangles in a scene are back facing


Backfaces


No backfaces

Backface Culling

- Convention:
  Triangle is front facing if vertices are ordered counterclockwise



Backface Culling

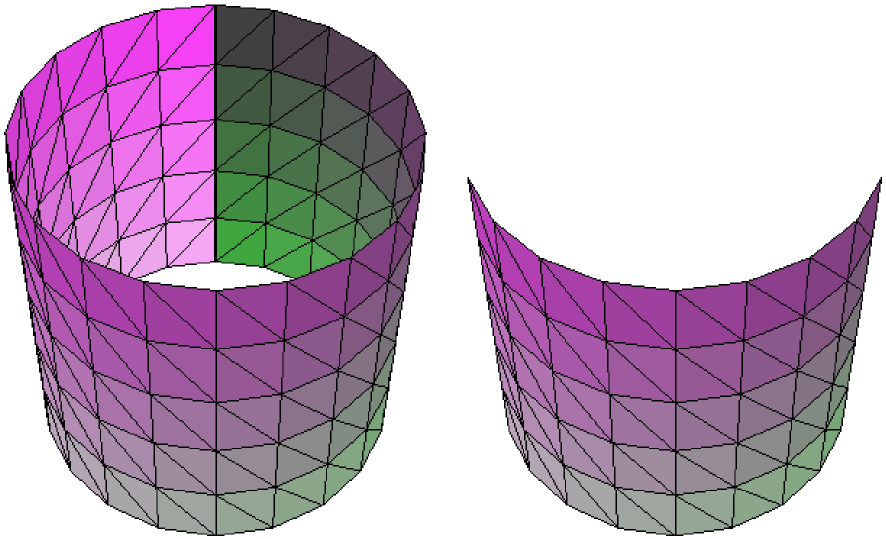
- Compute triangle normal after projection (homogeneous division)

$$\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$$

- Third component of $\mathbf{n}$ negative: front-facing, otherwise back-facing
  - Remember: projection matrix is such that homogeneous division flips sign of third component

OpenGL

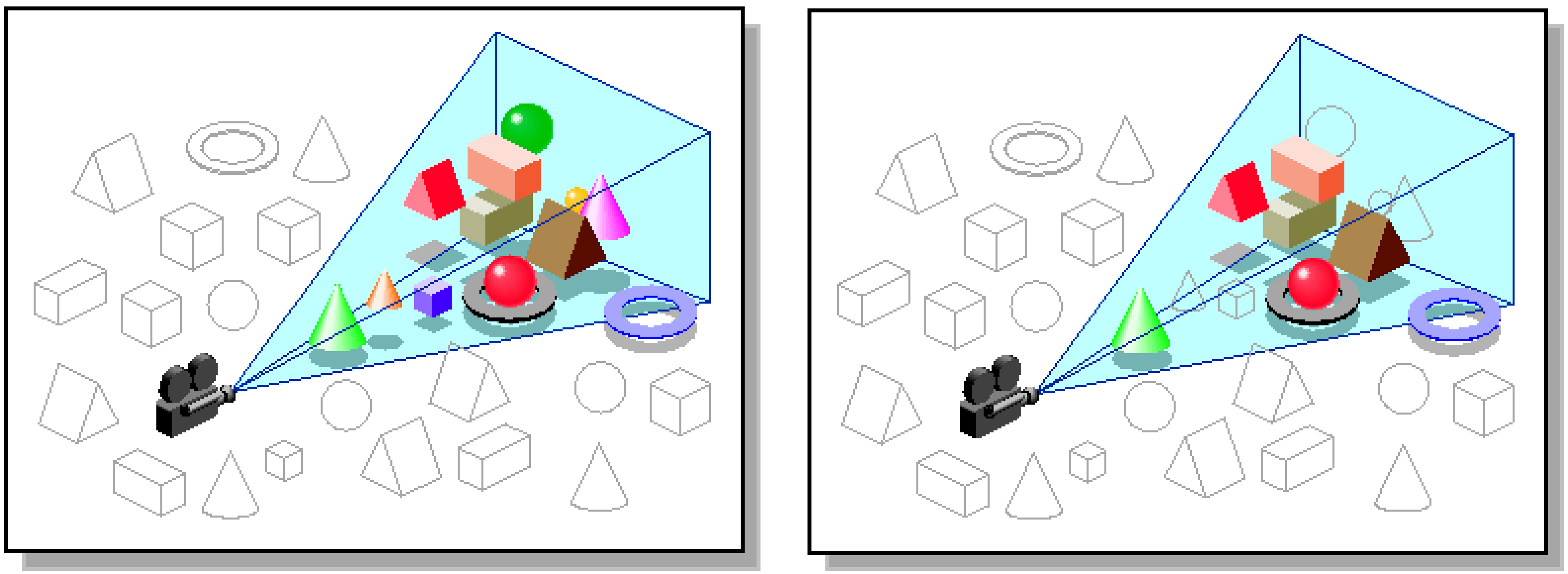
- OpenGL allows one- or two-sided triangles
  - One-sided triangles:
    glEnable(GL_CULL_FACE); glCullFace(GL_BACK)
  - Two-sided triangles (no backface culling):
    glDisable(GL_CULL_FACE)



glDisable(GL_CULL_FACE);   glEnable(GL_CULL_FACE);

Occlusion Culling

- Geometry hidden behind occluder cannot be seen
  - Many complex algorithms exist to identify occluded geometry



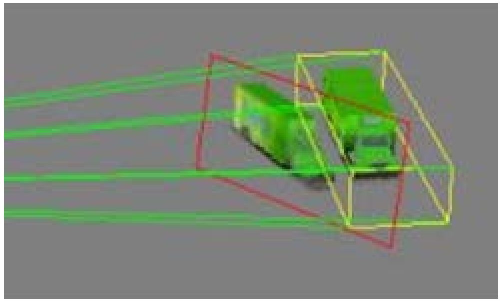
Images: SGI OpenGL Optimizer Programmer's Guide

Video

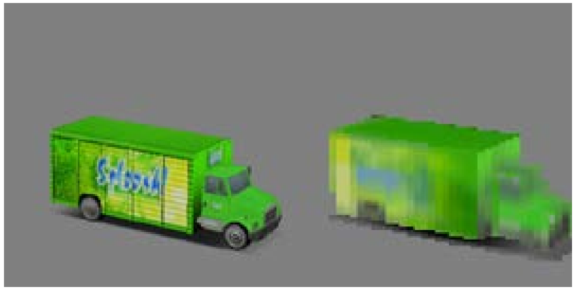
- Umbra 3 Occlusion Culling explained
  - http://www.youtube.com/watch?v=5h4QgDBwQhc

Level-of-Detail Techniques

- Don't draw objects smaller than a threshold
  - Small feature culling
  - Popping artifacts
- Replace 3D objects by 2D impostors
  - Textured planes representing the objects



Impostor generation



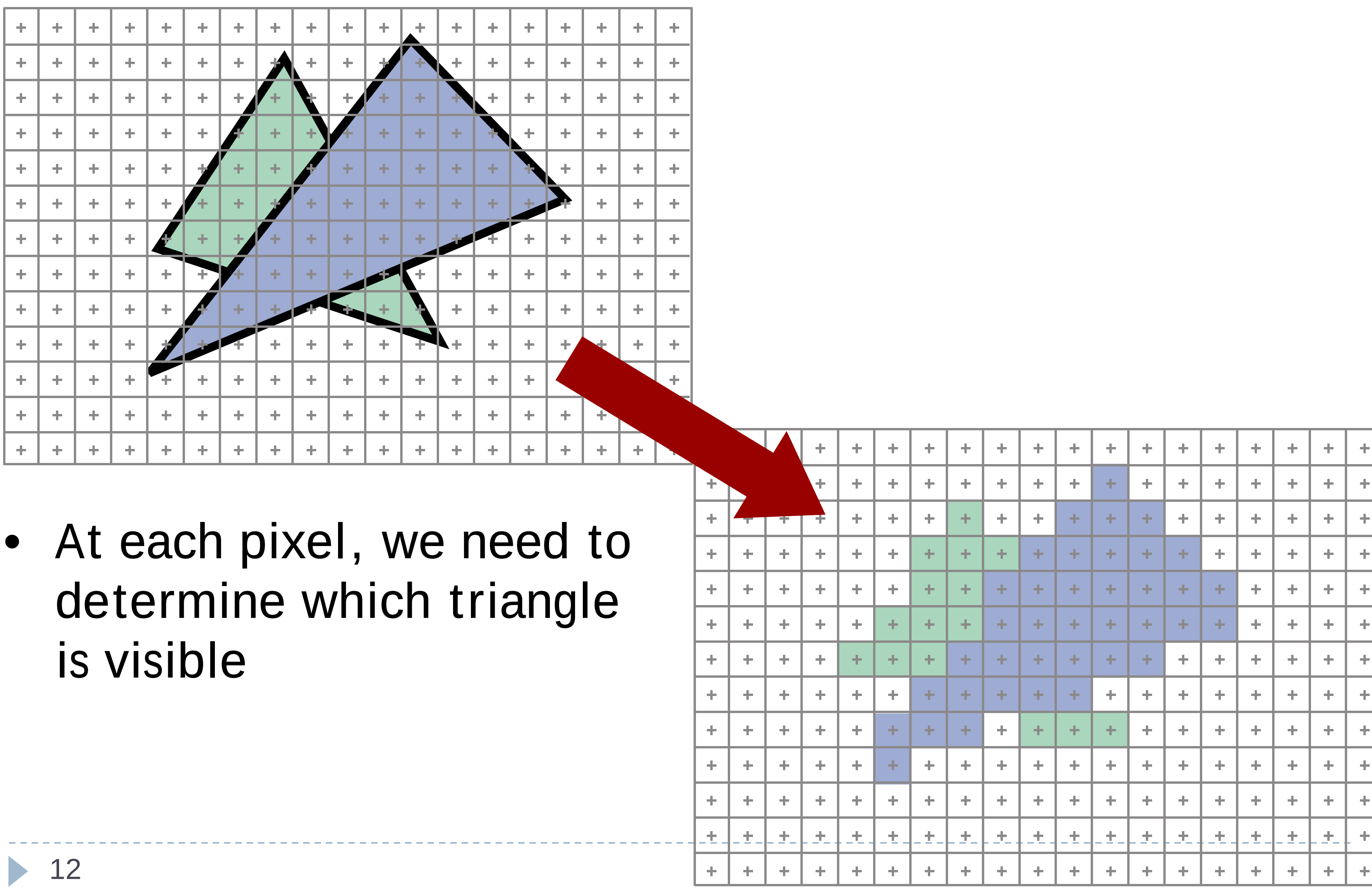
Original vs. impostor

- Adapt triangle count to projected size

Size dependent mesh reduction
(Data: Stanford Armadillo)

Occlusion

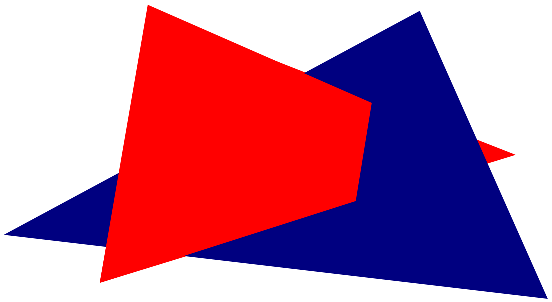
Occlusion



- At each pixel, we need to determine which triangle is visible

Painter's Algorithm

- Paint from back to front
- Need to sort geometry according to depth
- Every new pixel always paints over previous pixel in frame buffer
- May need to split triangles if they intersect



- Intuitive, but slow algorithm
- Still used today to render translucent geometry

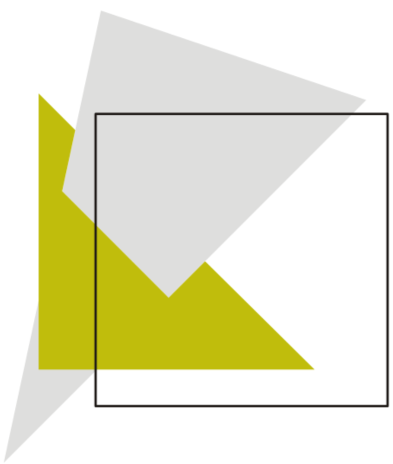
Z-Buffering

- Z-buffer stores depth (z-) value for each pixel
- Z-buffer is dedicated memory in GPU
- Algorithm:
  - Create z-buffer with as many entries as pixels in render window
  - Initialize z-buffer with farthest z value
  - During rasterization, compare stored value to new value
  - Update pixel only if new value is smaller

```
setpixel(int x, int y, color c, float z)
if(z < zbuffer(x,y)) then
{ zbuffer(x,y) = z; color(x,y) = c }
```

- Depth test is performed by GPU → very fast

Z-Buffer Example



∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

+

5	5	5	5	5	5	5
5	5	5	5	5	5	
5	5	5	5	5		
5	5	5	5			
5	5	5				
5	5					
5						

=

5	5	5	5	5	5	5	∞
5	5	5	5	5	5	∞	∞
5	5	5	5	5	∞	∞	∞
5	5	5	5	∞	∞	∞	∞
5	5	5	∞	∞	∞	∞	∞
5	5	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

5	5	5	5	5	5	5	∞
5	5	5	5	5	5	∞	∞
5	5	5	5	5	∞	∞	∞
5	5	5	5	∞	∞	∞	∞
5	5	5	∞	∞	∞	∞	∞
5	5	∞	∞	∞	∞	∞	∞
5	∞	∞	∞	∞	∞	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

+

7					
6	7				
5	6	7			
4	5	6	7		
3	4	5	6	7	
2	3	4	5	6	7

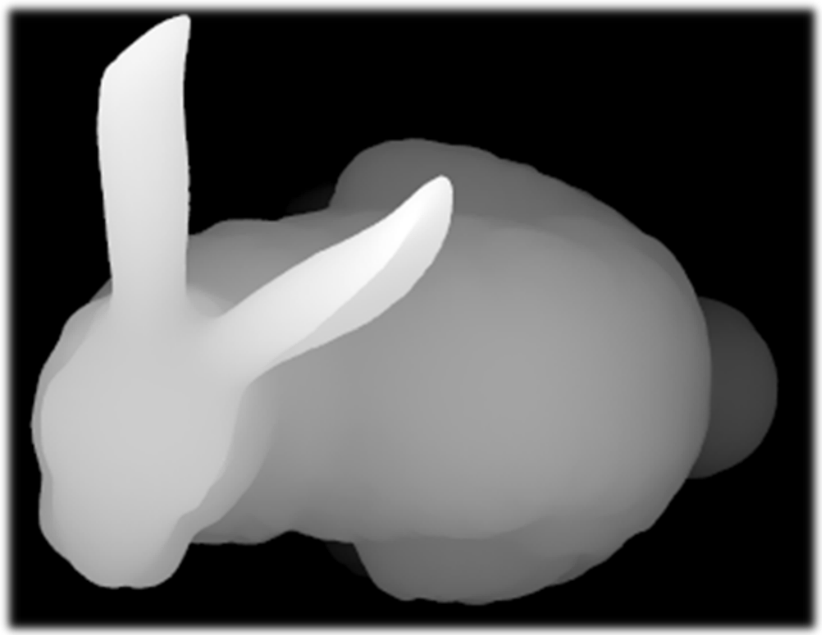
=

5	5	5	5	5	5	5	∞
5	5	5	5	5	5	∞	∞
5	5	5	5	5	∞	∞	∞
5	5	5	5	∞	∞	∞	∞
4	5	5	7	∞	∞	∞	∞
3	4	5	6	7	∞	∞	∞
2	3	4	5	6	7	∞	∞
∞	∞	∞	∞	∞	∞	∞	∞

Displaying the Z-Buffer

- Interpret z-buffer values as luminance values
- gl_FragCoord in fragment shader contains depth value
- Output this depth value as a color:

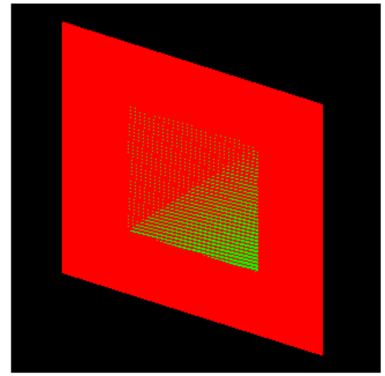
```
void main() { FragColor = vec4(vec3(gl_FragCoord.z), 1.0); }
```



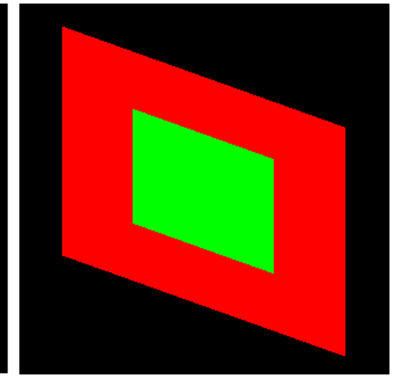
Z-Buffering in OpenGL

- In OpenGL applications:
  - Ask for a depth buffer when you create your GLFW window.
    - glfwOpenWindow(512, 512, 8, 8, 8, 0, **16**, 0, GLFW_WINDOW)
  - Place a call to glEnable(GL_DEPTH_TEST) in your program's initialization routine.
  - Set *zNear* and *zFar* clipping planes (glm::perspective(fovy, aspect, zNear, zFar)) to optimize depth buffer precision: near plane as far away as possible, far plane as close as possible without cutting into scene
  - Add GL_DEPTH_BUFFER_BIT parameter to glClear:
    - glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
- Z-buffer is non-linear: uses smaller depth bins in foreground for greater depth resolution near viewer

Z-Buffer Fighting



Z-buffer fighting

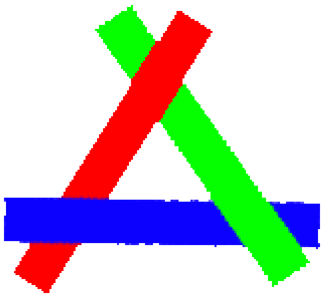


Desired result

- Problem: polygons close together don't get rendered correctly. Errors change with camera perspective → flicker
- Cause: differently colored fragments from different polygons being rasterized to same pixel and depth → not clear which is in front
- Solutions:
  - Move surfaces farther apart, so that fragments rasterize into different depth bins
  - Bring near and far planes closer together
  - Use a higher precision depth buffer. Note that OpenGL often defaults to 16 bit even if your graphics card supports 24 bit or 32 bit depth buffers

Translucent Geometry

- Need to depth sort translucent geometry and render with Painter's Algorithm (back to front)
- Problem: incorrect blending with cyclically overlapping geometry



- Solutions:
  - Back to front rendering of translucent geometry (Painter's Algorithm), after rendering opaque geometry
  - Theoretically: need to store multiple depth and color values per pixel (not practical in real-time graphics)