# Forward Rendering

Drawing things one at a time

Academy of Interactive Entertainment
www.aie.edu.au

Canberra Institute of Technology
Workplace • Online • Campus | cit.edu.au

# Render Pipeline So Far…

- Everything we've discussed so far is about how the GPU works when processing data

- How the hardware works is one thing, how you make use of it is a completely different story

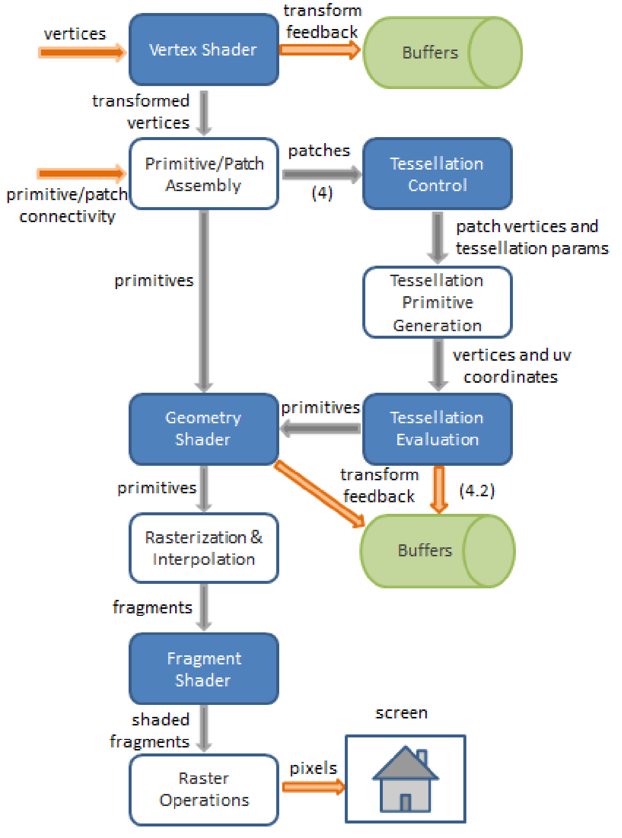
- The way in which your application renders its information is the most important factor when writing graphics-based applications

001011011010001010102

# Render Steps

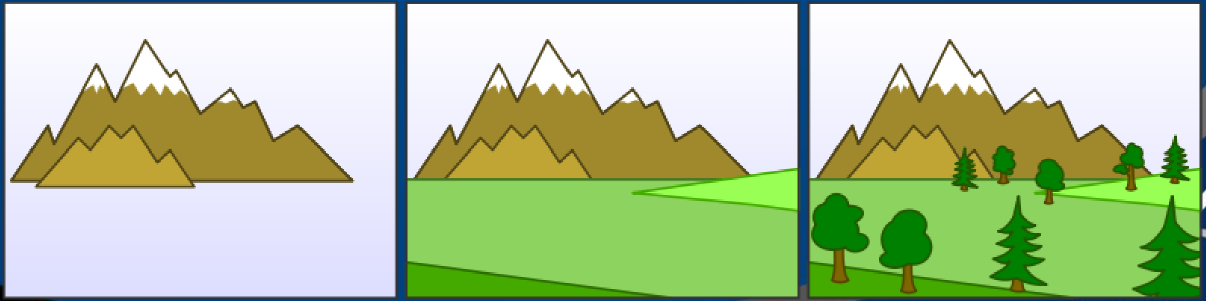
- So recapping the steps in rendering:
  - Start a draw call for a collection of primitives
  - Vertex Shader processes every vertex in the mesh independently from each other
  - The Tessellators processes all patches if it needs
  - The Geometry Shader then processes every primitive independently from each other
  - The Rasteriser then determines if a pixel should be rendered
  - The Fragment Shader then processes all pixels independently from each other
  - The Raster Operations perform any additional processing on the pixels
  - Repeat all steps for the next mesh being rendered…

- This process is referred to as Forward Rendering
  - In a future session we will discuss its opposite; Deferred Rendering



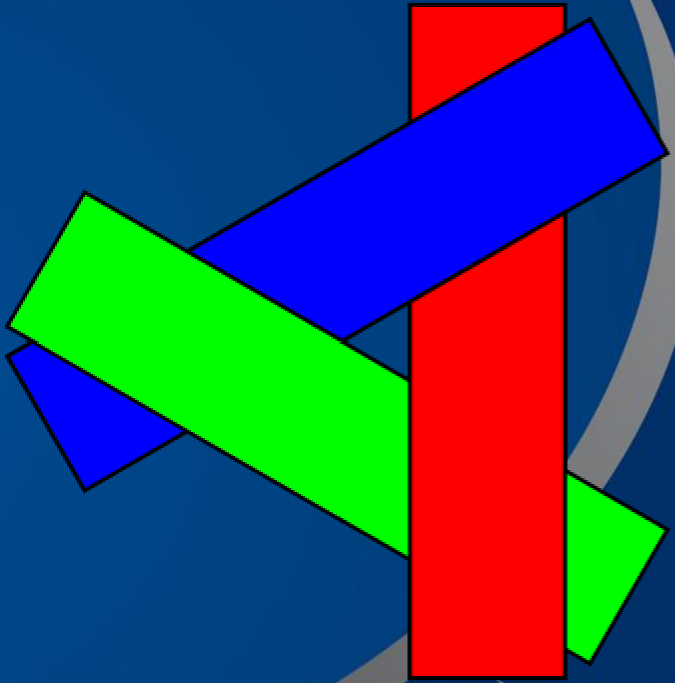
# Ordering – The Painter’s Algorithm

- One big problem exists in rendering
  - Visibility
- If I render a mesh, then render another mesh over it, which one should be visible?
  - One solution to this problem is called The Painter’s Algorithm
- The Painter’s Algorithim dictates that a scene should be drawn back to front, so that no distant object overlaps a closer object



# Painter’s Algorithm Flaws

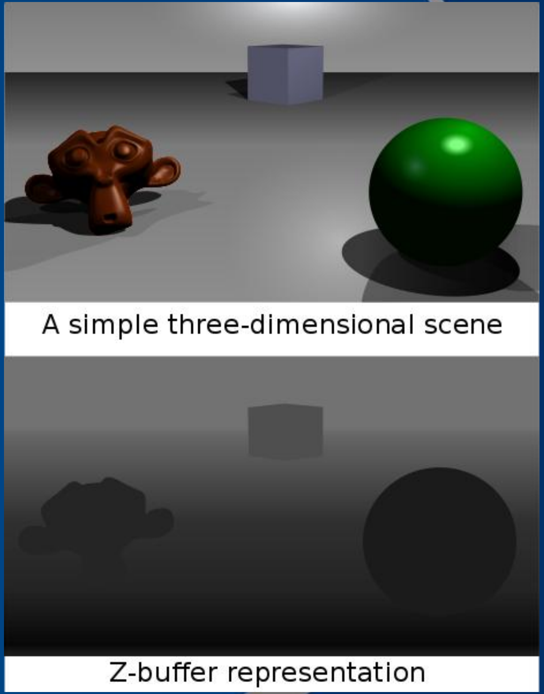
- Although being a great way to solve the issue, the algorithm still suffers from a flaw
  - How would it handle the following geometric arrangement?



- Also it means that thousands of pixels are being rendered multiple times in a single frame, even though only a single pixel colour will be visible in the end
  - In the previous example the pixels the trees are rendered into were already rendered as grass, which was a waste

# Enter the Z-buffer

- A solution to the render order problem was needed
  - The Z-Buffer was created to solve it
- The Z-Buffer can be thought of as a texture buffer, but instead of pixel colours it holds depth information at each pixel
- This depth information is then used by the Rasteriser
  - Typically any pixel currently being rendered will test its distance against the Z-buffer
  - If the pixel is “behind” the one already rendered then it is discarded

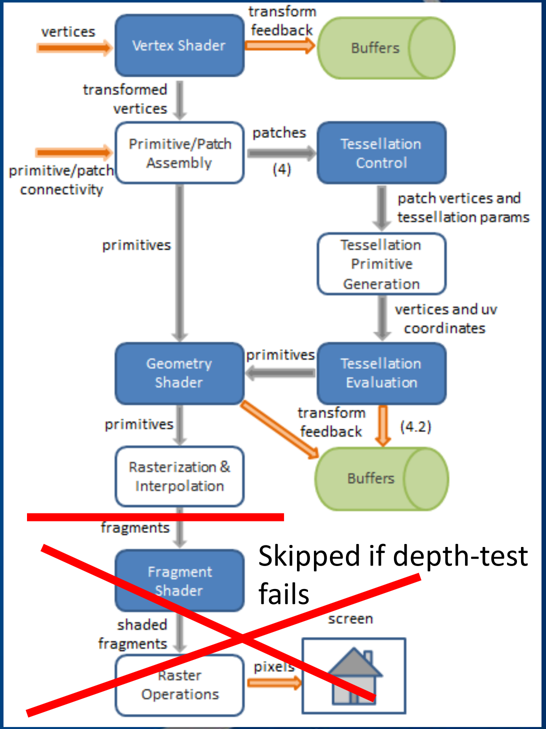


A simple three-dimensional scene

Z-buffer representation

# Front-to-Back Ordering

- Using the Z-Buffer we start a frame by clearing it to the furthest distance possible
  - To ensure everything we draw is “in front” of it
- We then render all the meshes and if a mesh renders into a pixel the depth is set
  - An object further away trying to render into the pixel will be caught by the Rasteriser and be ignored
- All geometry for a mesh will still go through the render pipeline but will end before the Fragment Shader
  - Modern shading algorithms are heavily pixel-based, so this can save considerable pixel processing time!
- Highly populated scenes can benefit from rendering the nearest objects first to reduce re-rendering pixels; a problem called Overdraw



Skipped if depth-test fails

Academy of Interactive Entertainment
www.aie.edu.au

Canberra Institute of Technology
Workplace • Online • Campus | cit.edu.au

# Accessing Depth

- With GLSL we can actually access a pixel's depth information from within the Fragment Shader

- There is a built-in global variable, **gl_FragCoord**
  - A **vec4**
  - X and Y refer to the window coordinates of the current pixel fragment
  - Z refers to the depth
  - W actually stores 1/W which was used during the projection from 3D space to 2D window coordinates

# Summary

- Render order matters!
- A Z-buffer takes care of culling any pixels being drawn behind already drawn pixels
  - Must remember to “wipe” the z-buffer each frame
  - Not clearing the z-buffer can cause some interesting side effects

http://en.wikipedia.org/wiki/Painter's_algorithm
http://en.wikipedia.org/wiki/Z-buffering