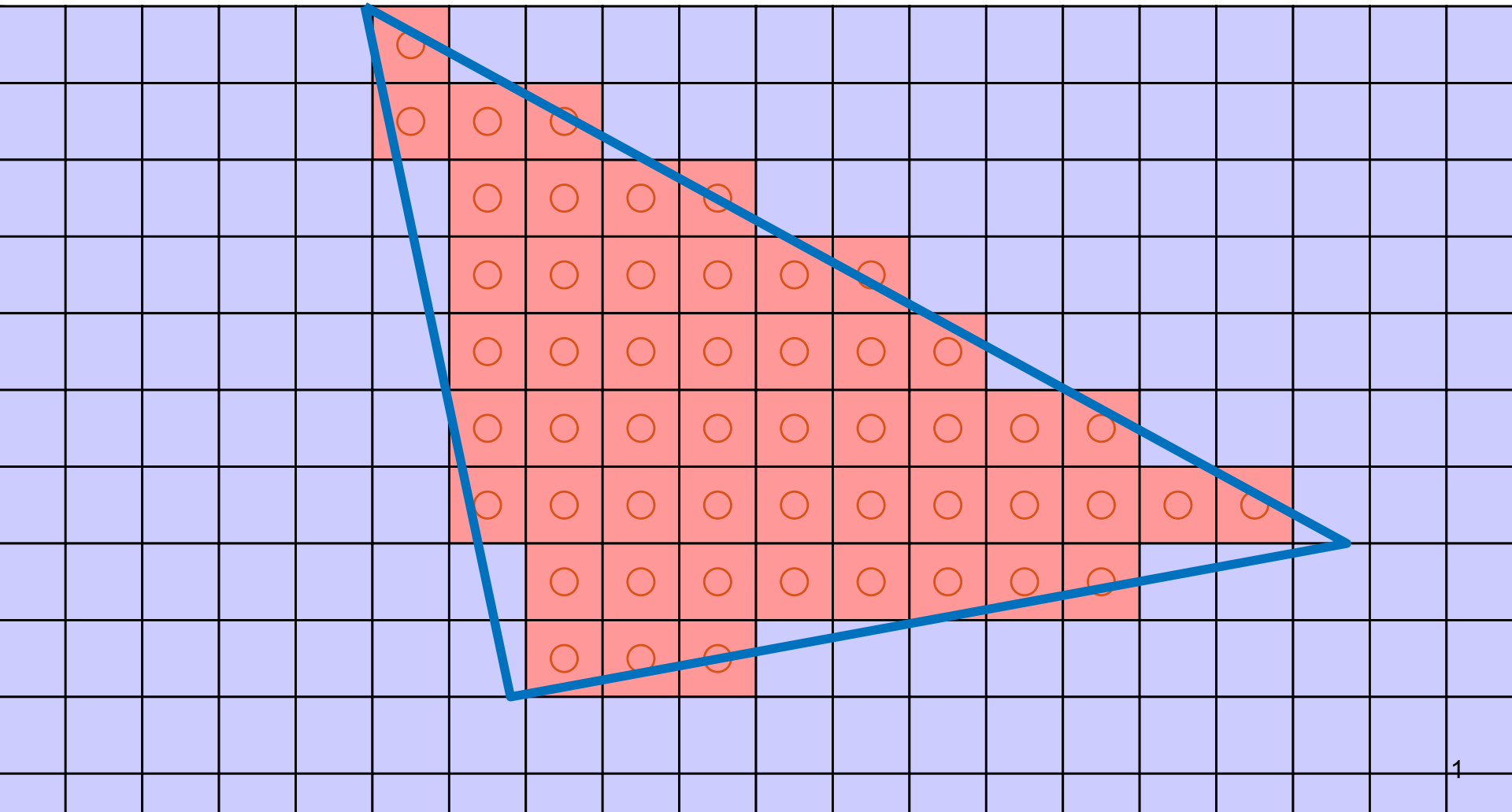
Rasterization & The Graphics Pipeline

15.1 Rasterization Basics

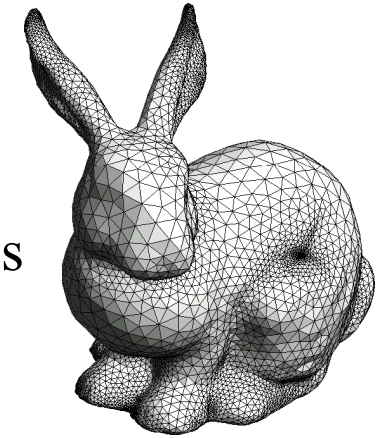


In This Video

- The Graphics Pipeline and how it processes triangles
  - Projection, rasterisation, shading, depth testing
- DirectX12 and its stages

Modern Graphics Pipeline

- Input
  - Geometric model
    - Triangle vertices, vertex normals, texture coordinates
  - Lighting/material model (shader)
    - Light source positions, colors, intensities, etc.
    - Texture maps, specular/diffuse coefficients, etc.
  - Viewpoint + projection plane
  - You know this, you've done it!

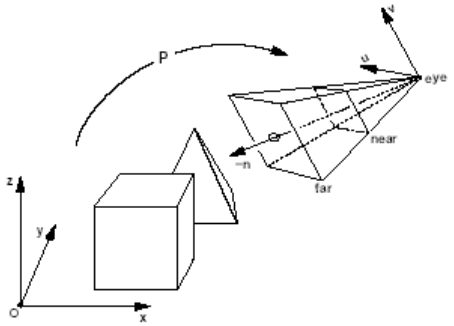


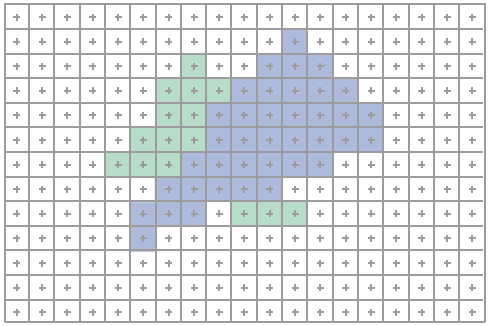
- Output
  - Color (+depth) per pixel

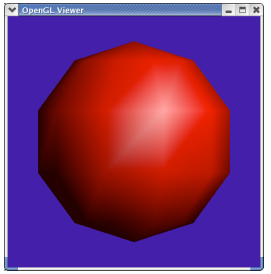
Colbert & Krivanek

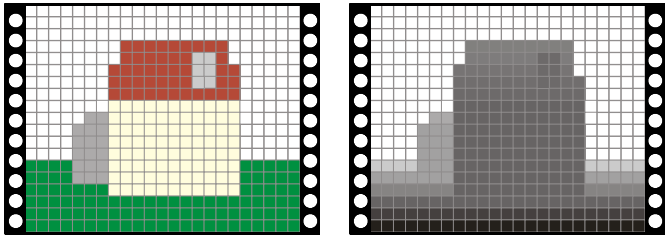
The Graphics Pipeline

- Project vertices to 2D (image)
- Rasterize triangle: find which pixels should be lit
- Compute per-pixel color
- Test visibility (Z-buffer), update frame buffer color



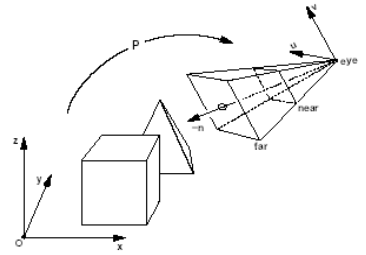




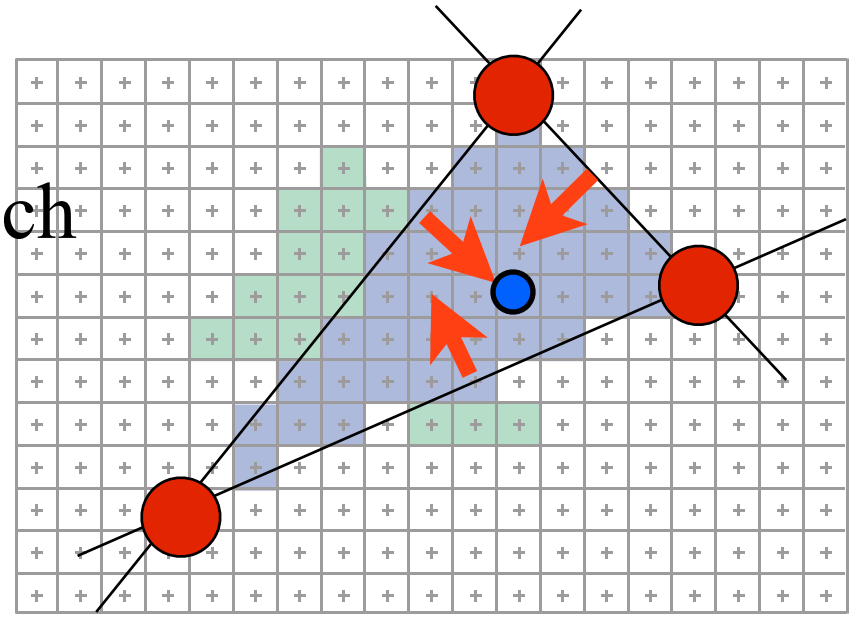


The Graphics Pipeline

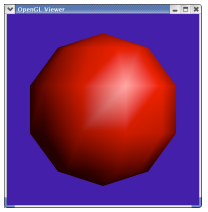
- Project vertices to 2D (image)

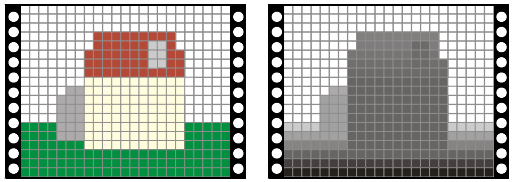


- Rasterize triangle: find which pixels should be lit
  - For each pixel, test 3 edge equations
    - if all pass, draw pixel

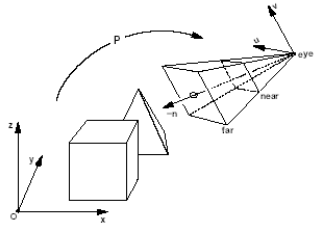


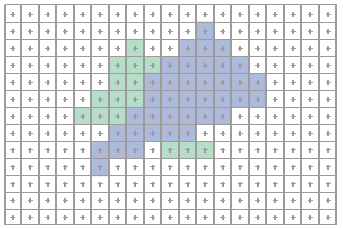
- Compute per-pixel color
- Test visibility (Z-buffer), update frame buffer color

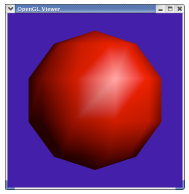




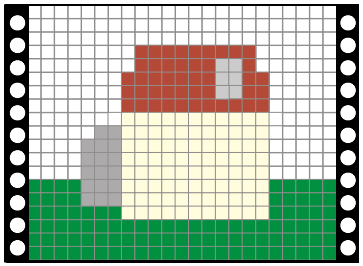
The Graphics Pipeline



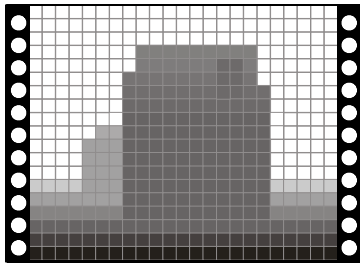




- Perform projection of vertices
- Rasterize triangle: find which pixels should be lit
- Compute per-pixel color
- Test visibility, update frame buffer color
  - Store minimum distance to camera for each pixel in "Z-buffer"
    - ~same as t_{min} in ray casting!
  - if new_z < zbuffer[x,y]
    zbuffer[x,y]=new_z
    framebuffer[x,y]=new_color



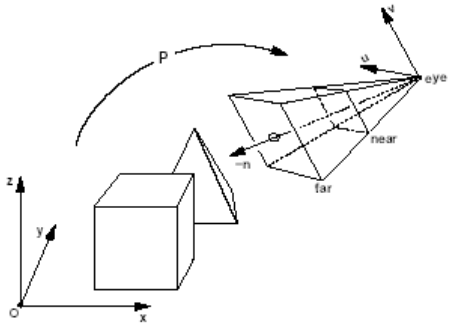
frame buffer

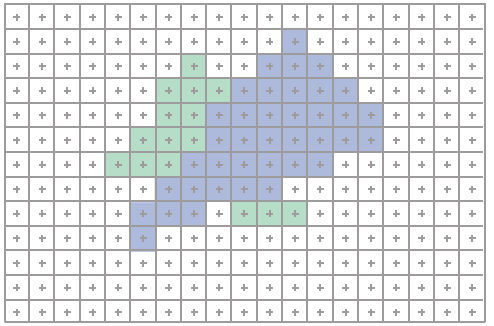


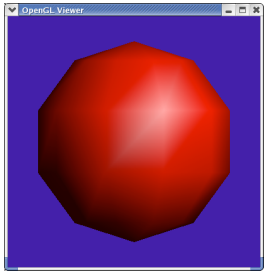
Z buffer

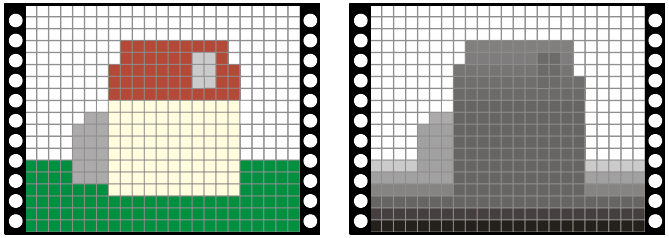
The Graphics Pipeline

```
For each triangle
  transform into eye space
  (perform projection)
  setup 3 edge equations
  for each pixel x,y
    if passes all edge equations
      compute z
      if z<zbuffer[x,y]
        zbuffer[x,y]=z
        framebuffer[x,y]=shade()
```





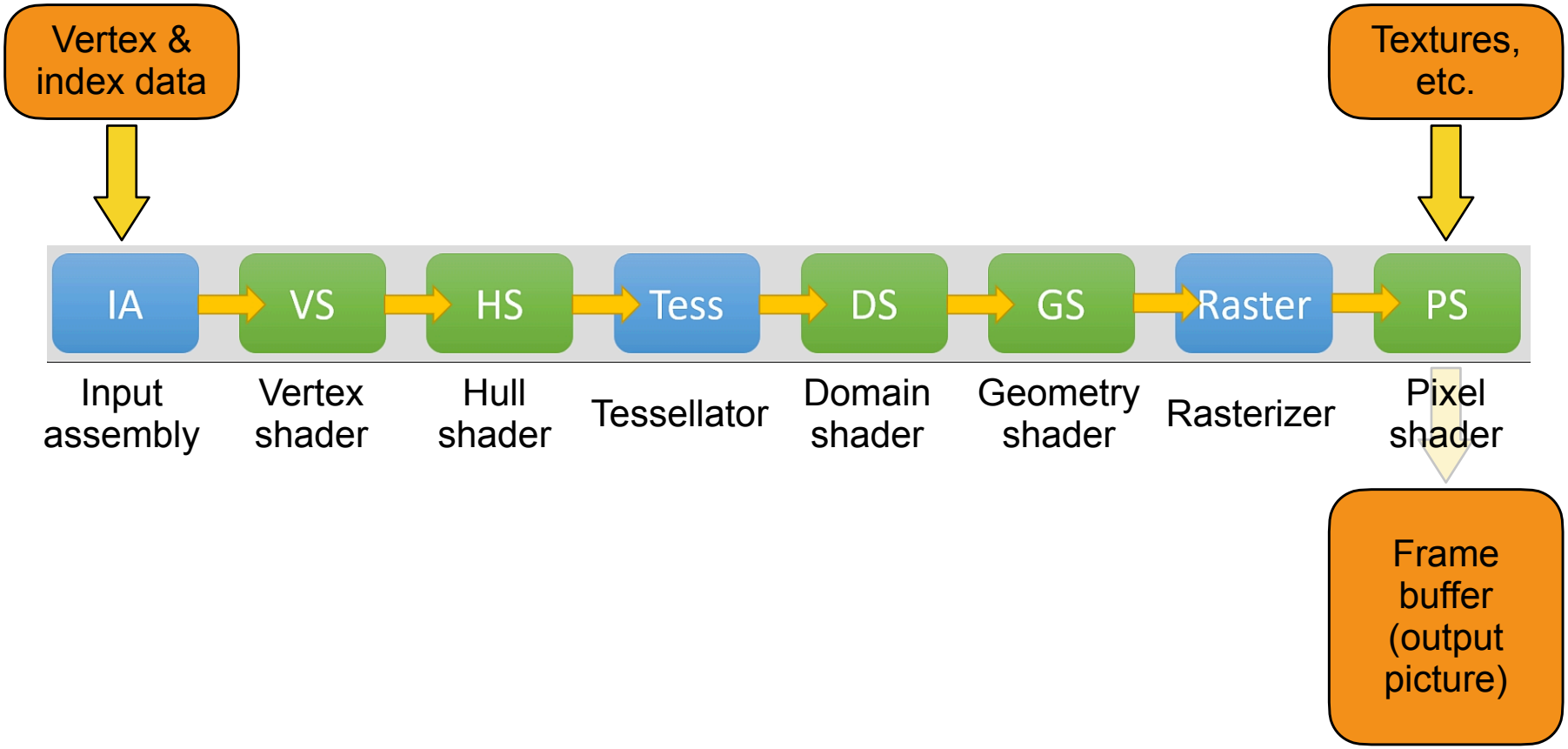




DirectX 12 Pipeline

(Simplified version)

(Vulkan & Metal are highly similar)



Fixed-function hardware | Programmable stage | Memory input/output

DirectX 12 Pipeline

(Simplified version)

1. Project vertices to 2D (image)

Fixed-function hardware | Programmable stage | Memory input/output

DirectX 12 Pipeline

(Simplified version)

1. Project vertices to 2D (image)
2. Figure out which pixels to draw

Fixed-function hardware | Programmable stage | Memory input/output

DirectX 12 Pipeline

(Simplified version)

1. Project vertices to 2D (image)
2. Figure out which pixels to draw
3. Compute per-pixel color

Fixed-function hardware | Programmable stage | Memory input/output

DirectX 12 Pipeline

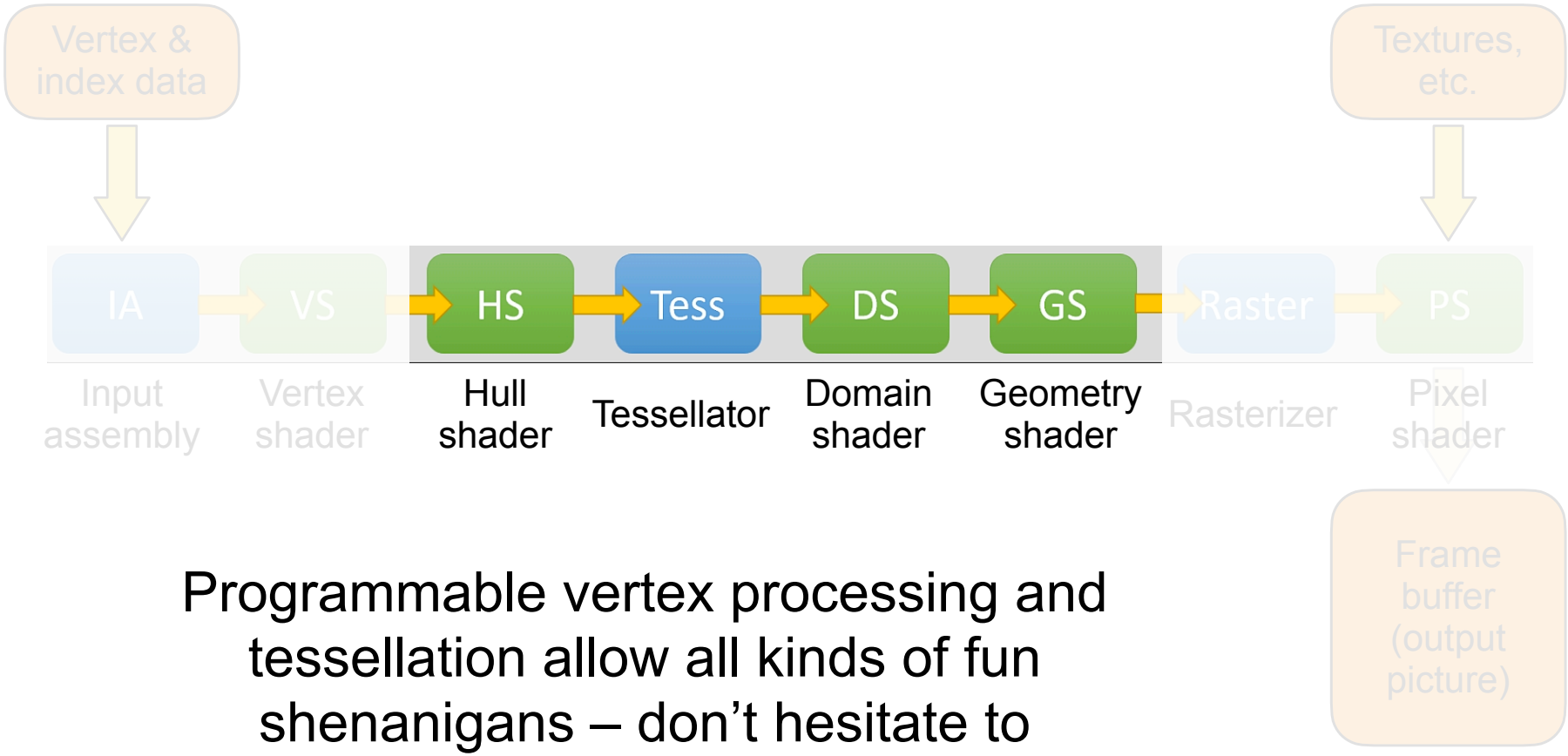
(Simplified version)

1. Project vertices to 2D (image)
2. Figure out which pixels to draw
3. Compute per-pixel color
4. Test depth and update frame buffer (“ROP”)

Fixed-function hardware | Programmable stage | Memory input/output

DirectX 12 Pipeline

(Simplified version)



Programmable vertex processing and tessellation allow all kinds of fun shenanigans – don't hesitate to explore for extra credit!

Fixed-function hardware | Programmable stage | Memory input/output