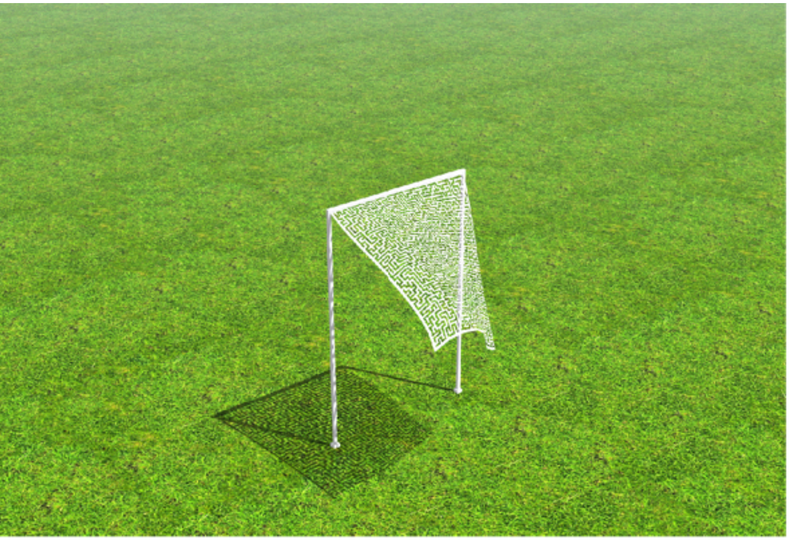
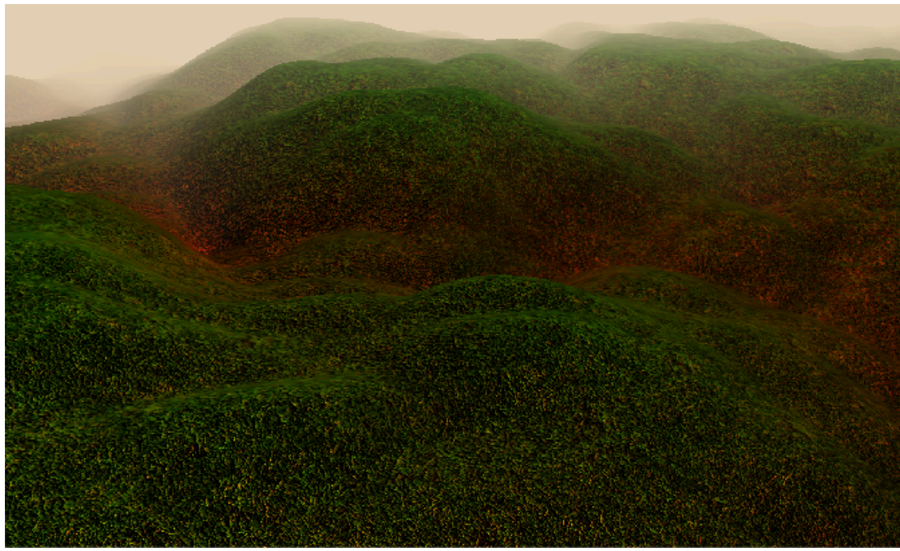
COSC342: Computer Graphics

2017





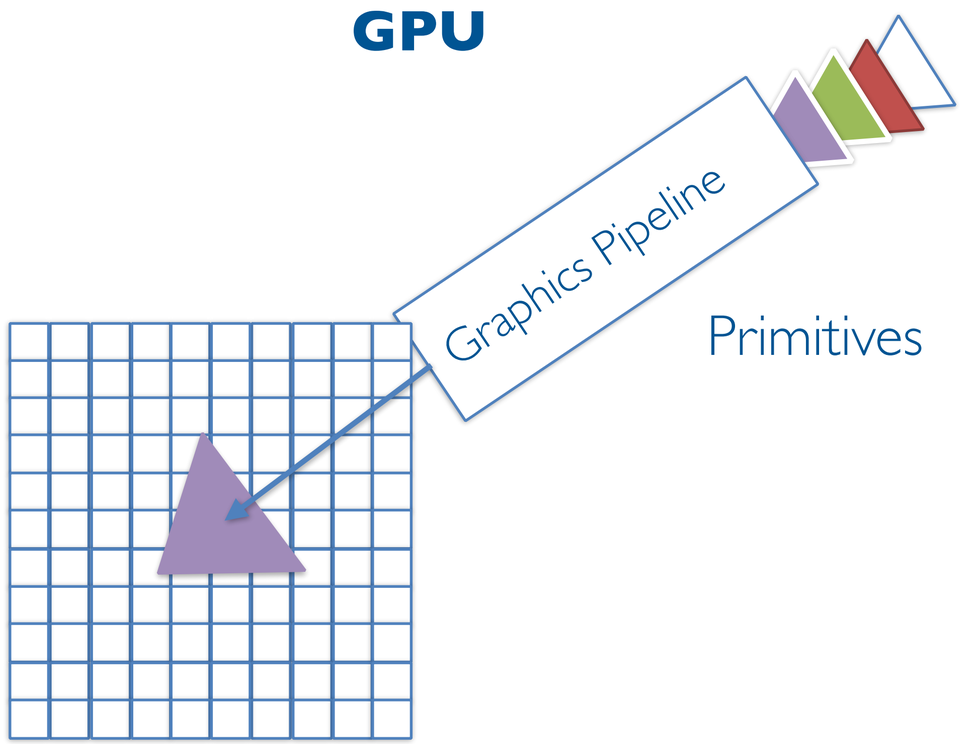
Tutorial 12

RECAP GRAPHICS PIPELINE

Stefanie Zollmann

GPUS DO RASTERISATION

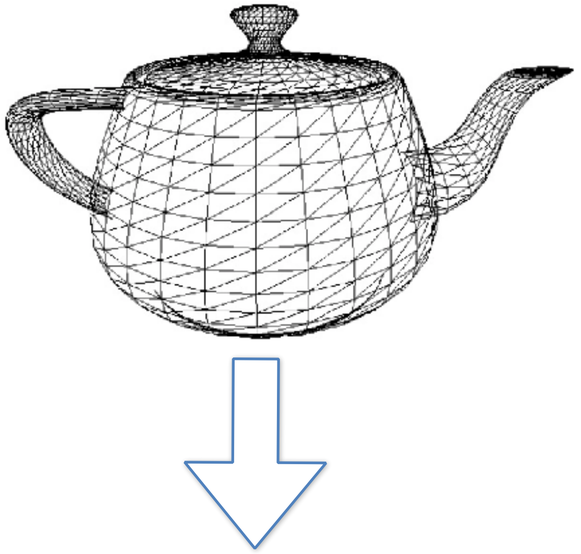
- **Rasterisation**
  - Computing for each triangle which pixel it covers
  - Triangle centric approach
  - Rasteriser only needs one triangle at a time



For each triangle
For each pixel
Does triangle cover pixel
Keep closest hit

GRAPHICS PIPELINE

- **Input**
  - Geometric model
    - Vertices, normals, texture coordinates
  - Lighting/shading model
    - Light positions
    - View point and virtual camera configuration
- **Output**
  - Colour (and depth) per pixel on a screen

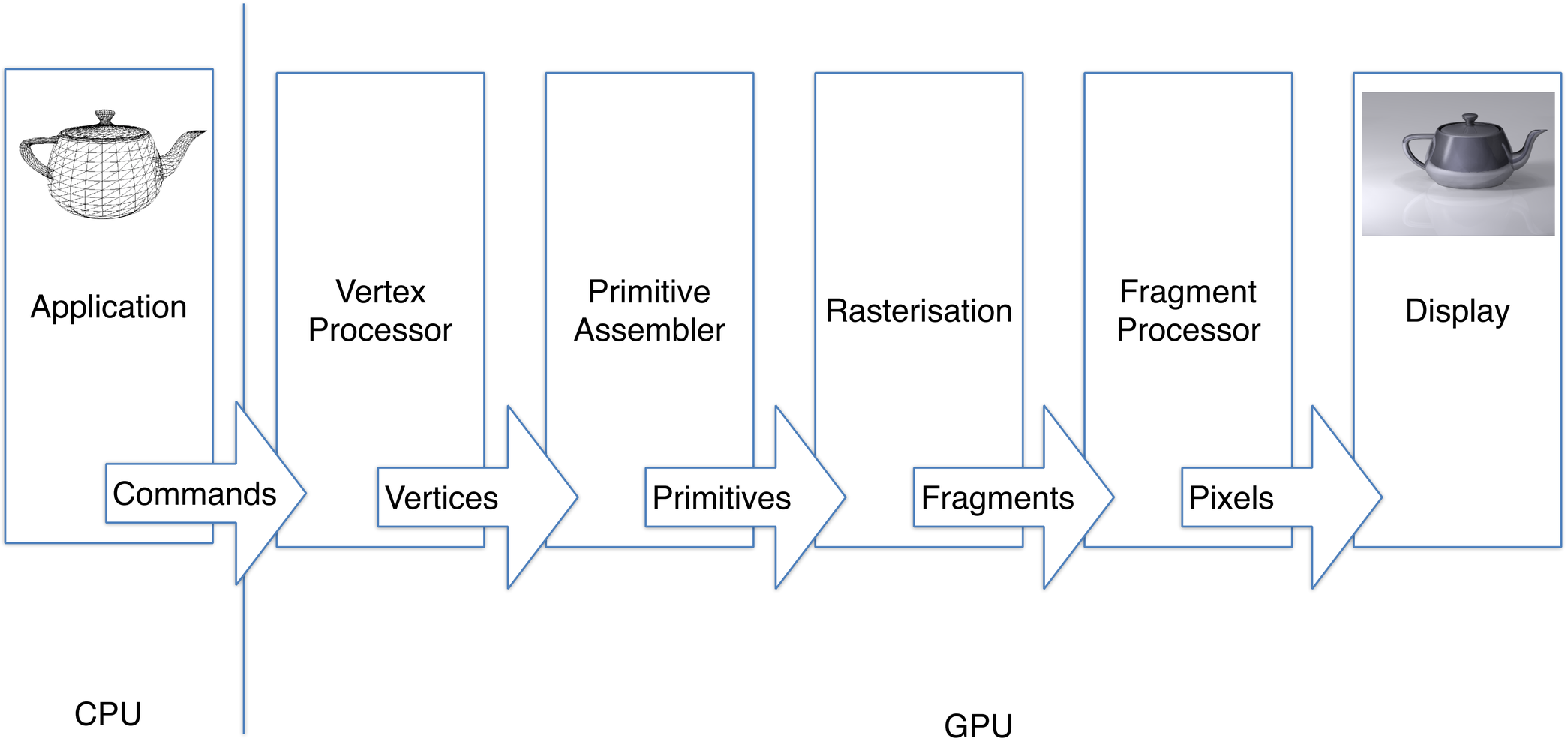


https://commons.wikimedia.org/wiki/File:Utah_teapot_simple.pn

INPUT GEOMETRY

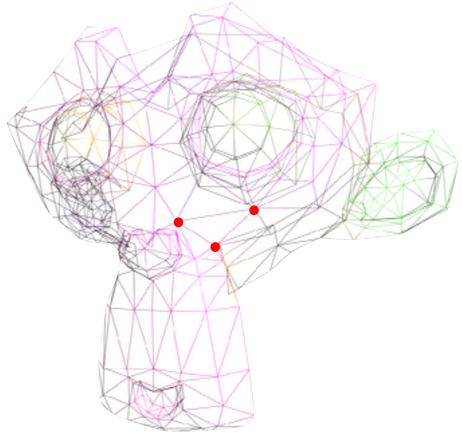
```
// 1rst attribute buffer : vertices
    glEnableVertexAttribArray(0);
    glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
    glVertexAttribPointer(
                          0,                  // attribute 0. No particular
reason for 0, but must match the layout in the shader.
                          3,                  // size
                          GL_FLOAT,           // type
                          GL_FALSE,           // normalized?
                          0,                  // stride
                          (void*)0            // array buffer offset
                          );

    // Draw the triangle !
    glDrawArrays(GL_TRIANGLES, 0, 3); // 3 indices starting at 0 -> 1 triangle
```

GRAPHICS PIPELINE



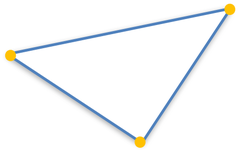
GRAPHICS PIPELINE

3D mesh

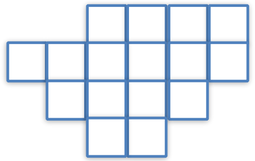


Vertices in screen space

Triangles in screen space



Fragments



Fragments with colours

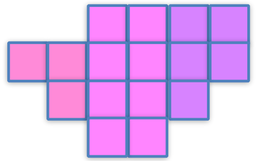
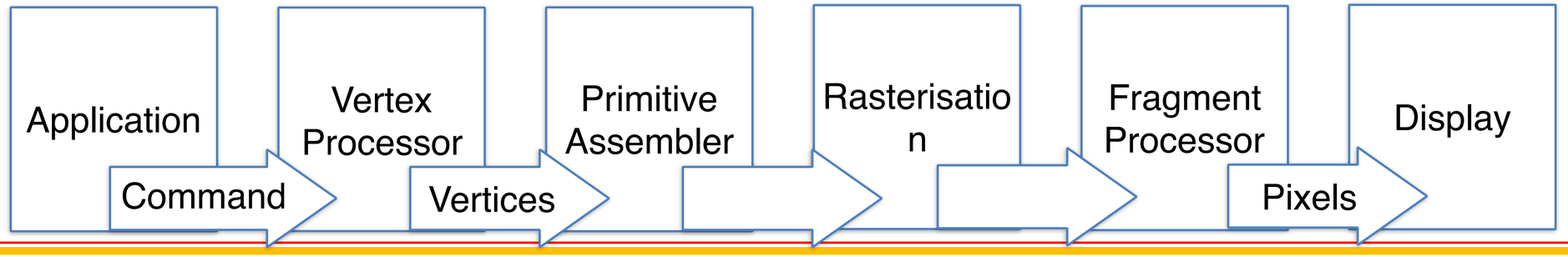
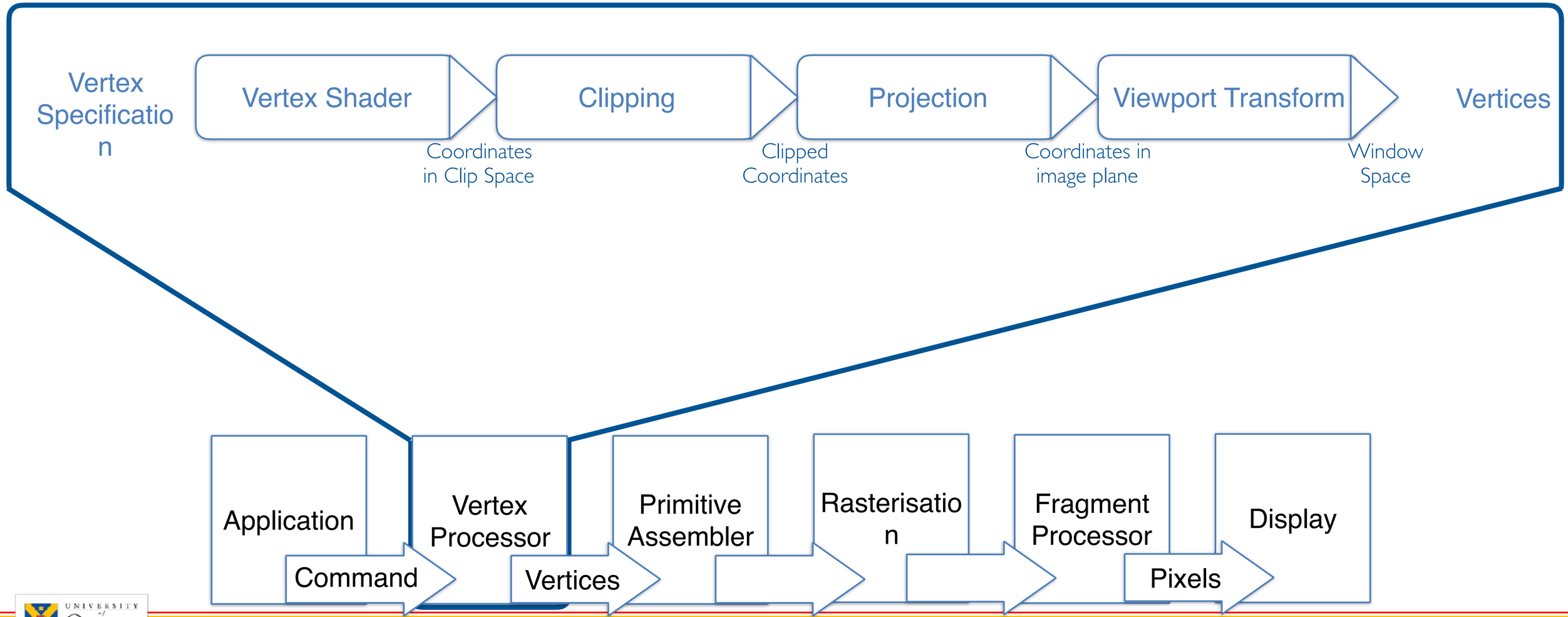


Image Output

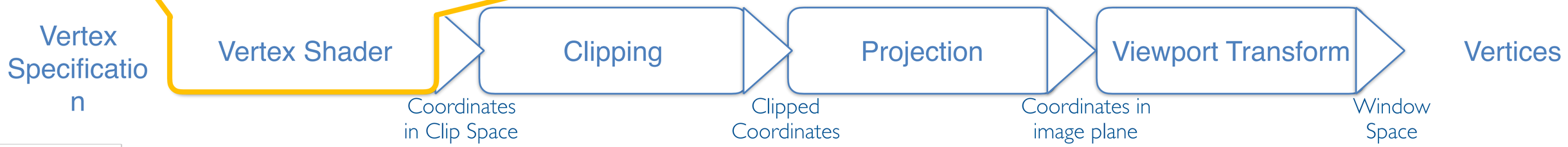


VERTEX PROCESSING

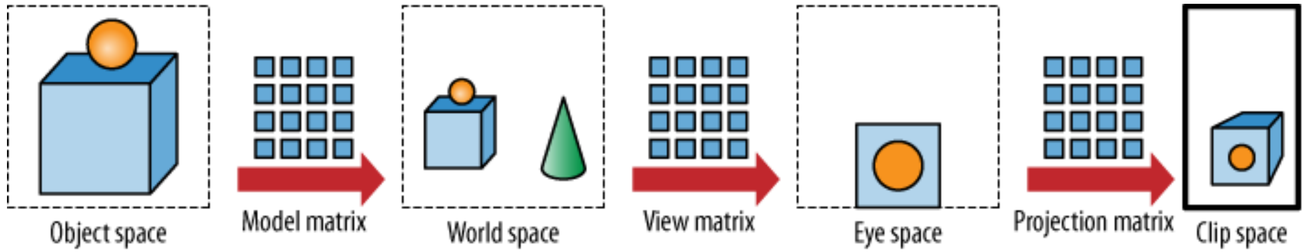


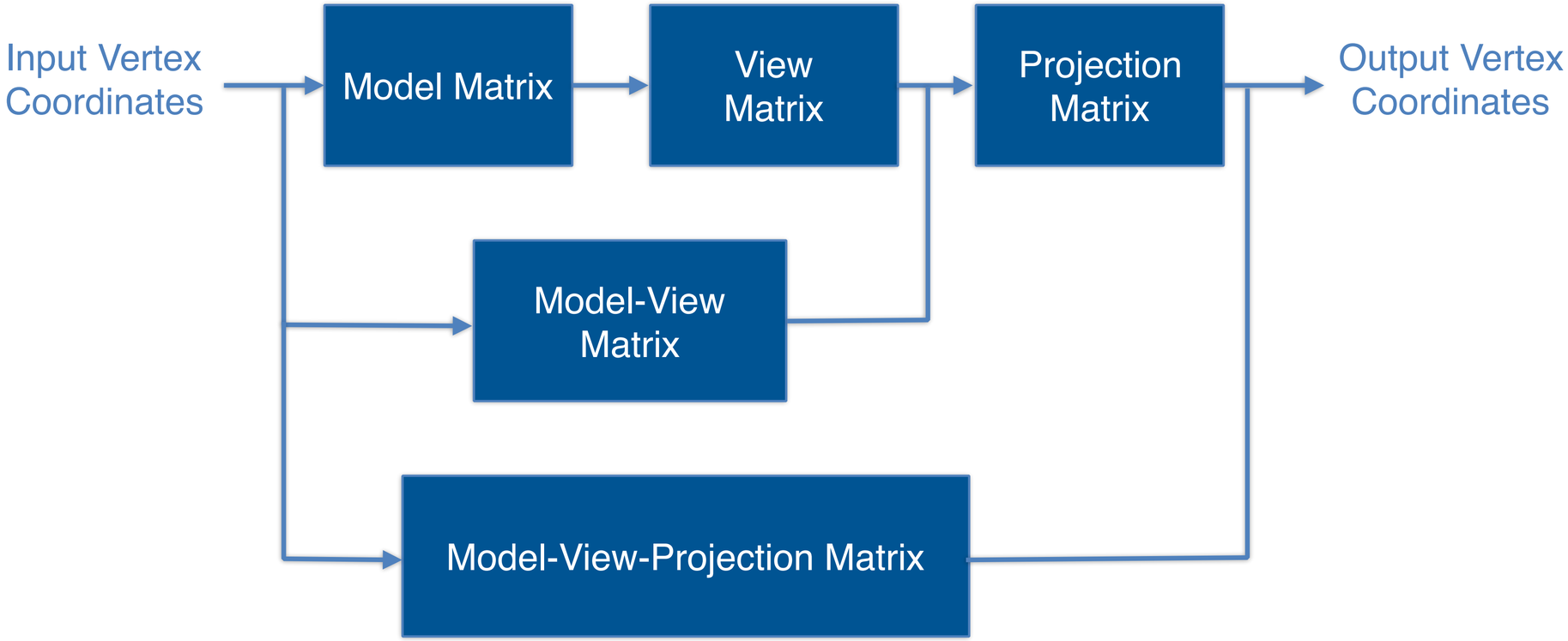
VERTEX SHADER

- Programmable shader stage (GPU program)
- "Shaders" were small programs performing lighting calculations
- Transforms input vertex stream into stream of vertices mapped onto the screen (clip space coordinates: homogeneous coordinates)
- Uses model, view and projection matrices to transform from model to world to view and to clip space

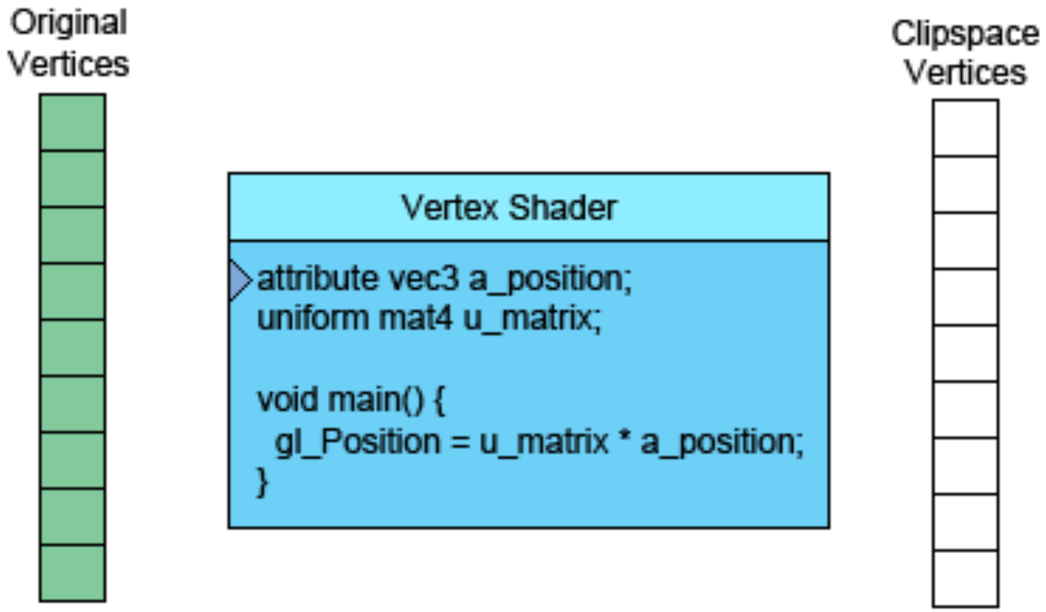


VERTEX SHADER: TRANSFORMATIONS



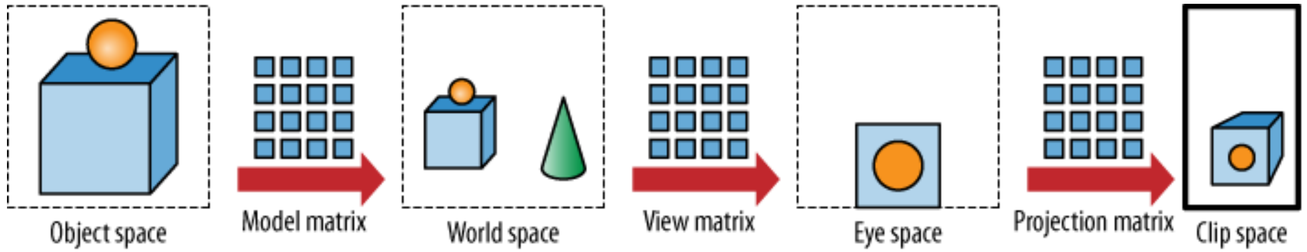


VERTEX SHADER



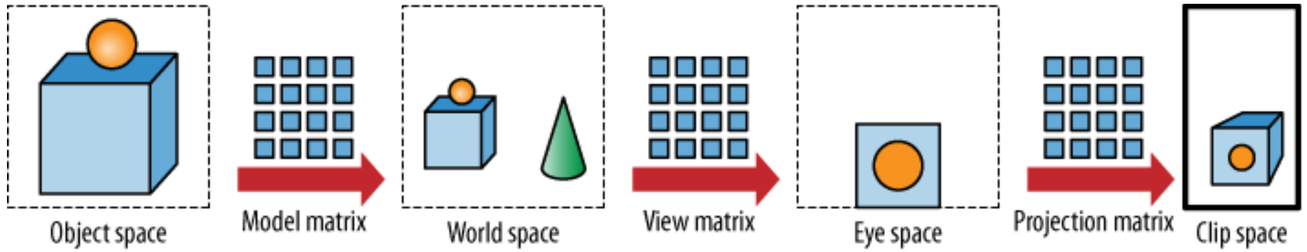
https://webglfundamentals.org/webgl/lessons/webgl-how-it-works.html

VERTEX SHADER: EXAMPLE




```
// Input vertex data, different for all executions of this shader.
layout(location = 0) in vec3 vertexPosModelspace;

// Output data ; will be interpolated for each fragment.
out vec3 posWorldspace;

// Values that stay constant for the whole mesh.
// Model-view-projection matrix
uniform mat4 MVP;

void main(){
  // Output position of the vertex, in clip space : MVP * position
  gl_Position =  MVP * vec4(vertexPosModelspace,1);
}
```

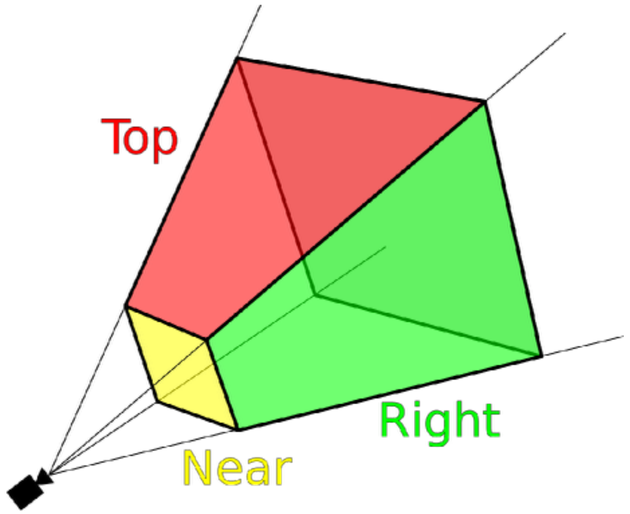
VERTEX SHADER: EXAMPLE




```
// Build the model matrix -get from object
   glm::mat4 ModelMatrix = this->getTransform();
   glm::mat4 MVP = camera->getViewProjectionMatrix() * ModelMatrix;
   // Send our transformation to the currently bound shader,
   // in the "MVP" uniform
   shader->updateMVP(MVP);
```

CLIPPING

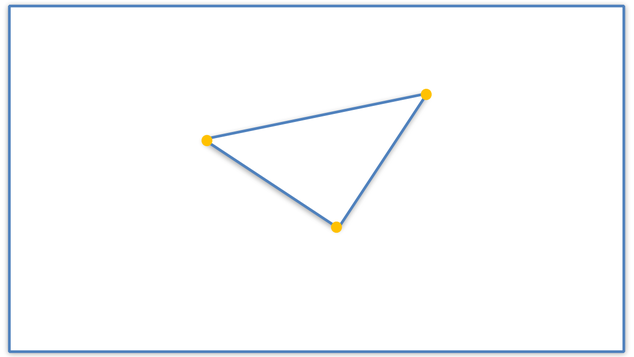
- Coordinates not entirely in view are clipped to the view volume



By MithrandirMage [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons

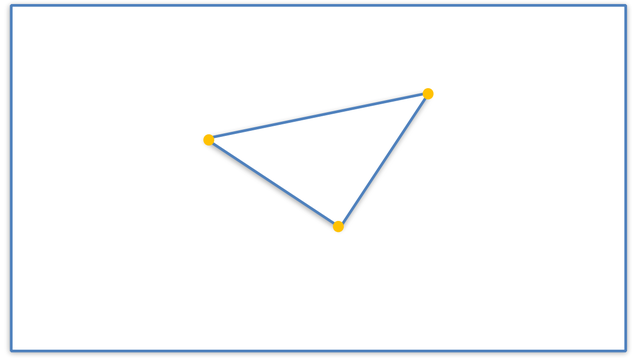
PROJECTION

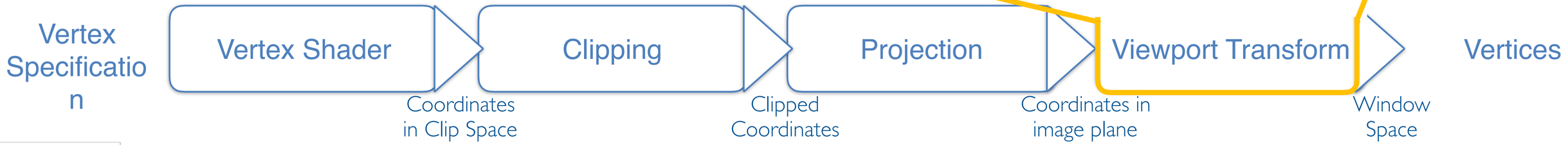
- We need normalised device coordinates ranging from -1 to +1 on each axis regardless of the shape or size of the actual screen
- Transforms clip space coordinates into normalised device coordinates (NDC)
- Divide clip space coordinates (x,y,z) by clip-space factor w (x/w, y/w, z/w)



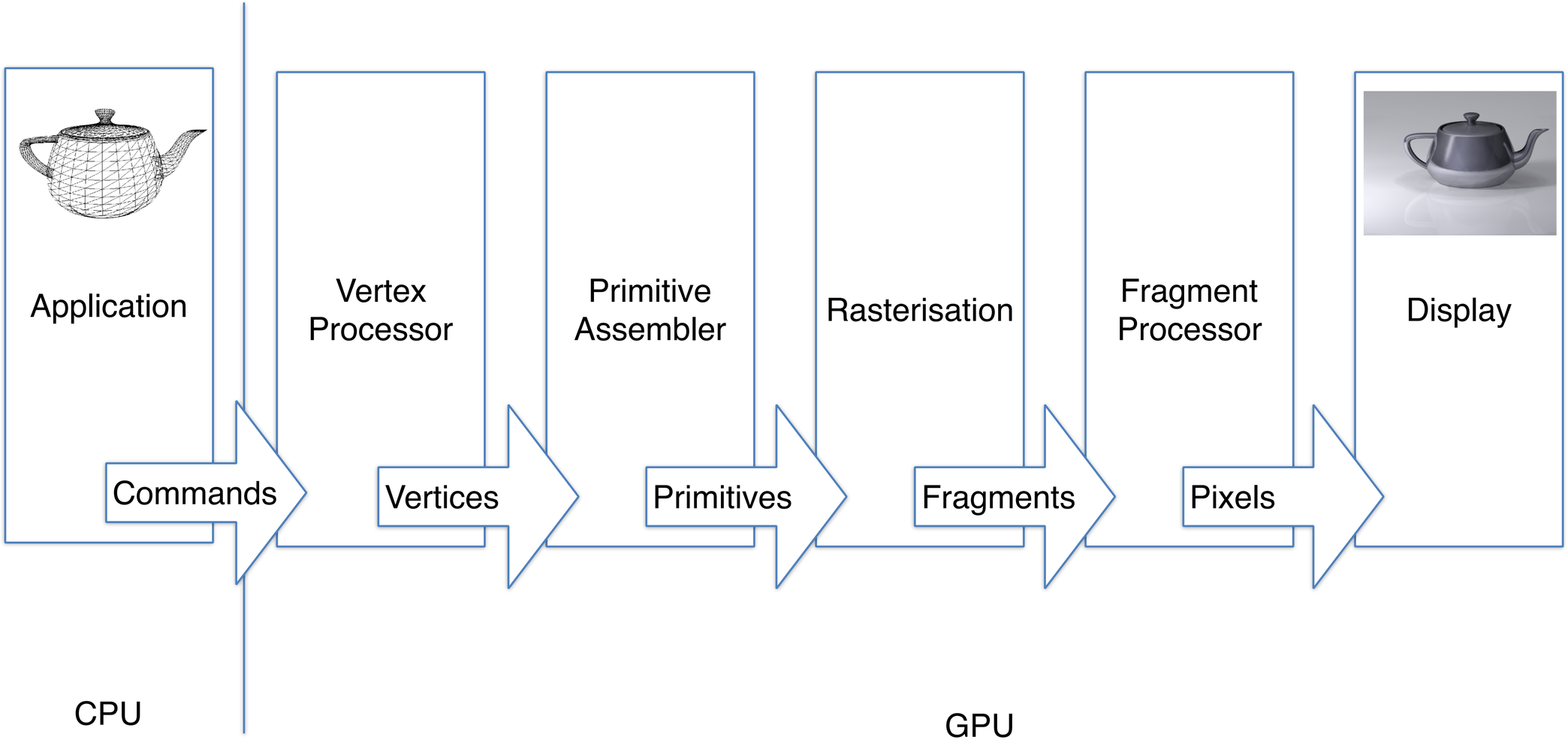
VIEWPORT TRANSFORM

- Map resolution-independent normalised device coordinates to a rectangular window in the frame buffer, the viewport
- Is defined by the viewport definition
- Output in window (pixel) coordinates

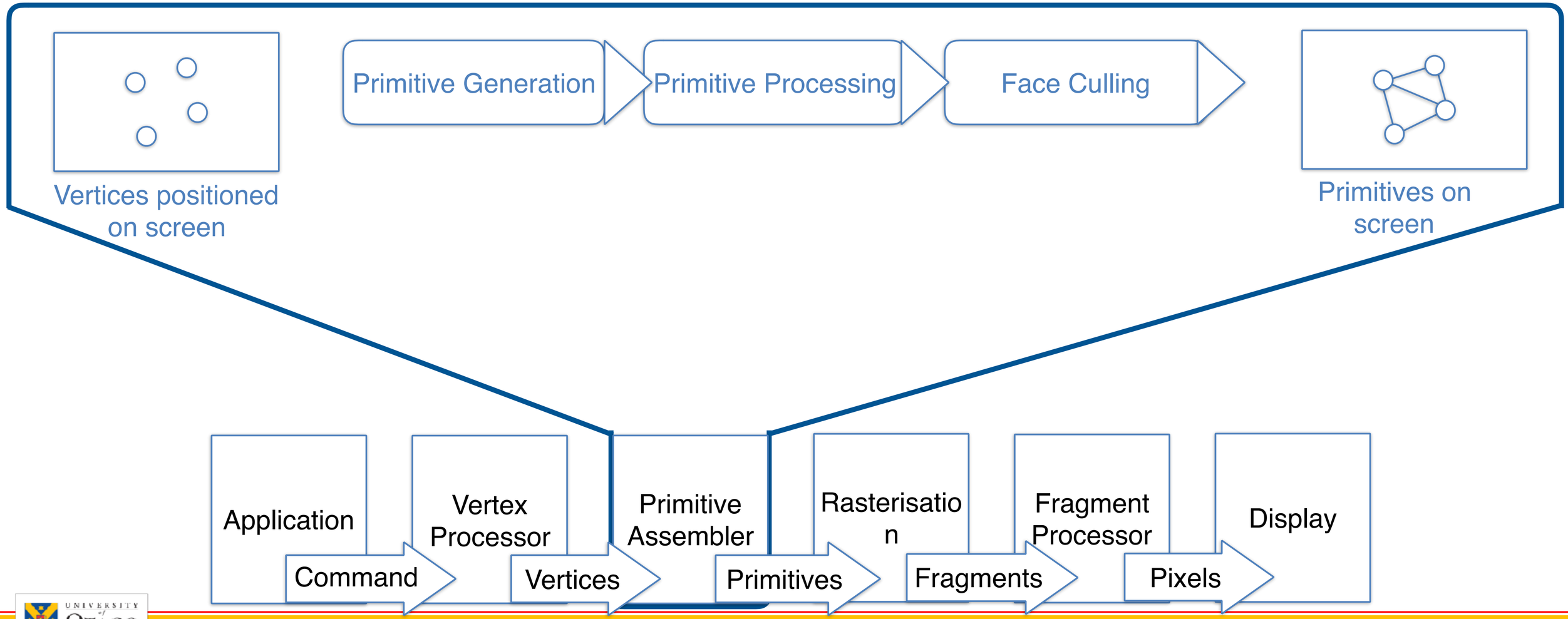




GRAPHICS PIPELINE



PRIMITIVES ASSEMBLY



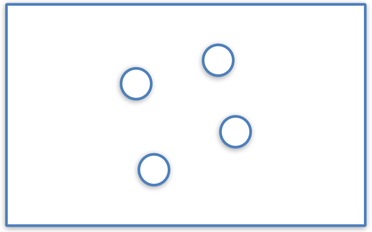
PRIMITIVES ASSEMBLY

Vertices positioned on screen

Primitives on screen

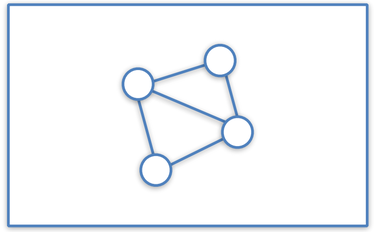
- Primitive assembly receives processed vertices, and the vertex connectivity information as input
- Divides them into a sequence of individual base primitives
- Primitives now have a certain facing
- Face culling discards faces based on their facing

PRIMITIVES ASSEMBLY: PRIMITIVES

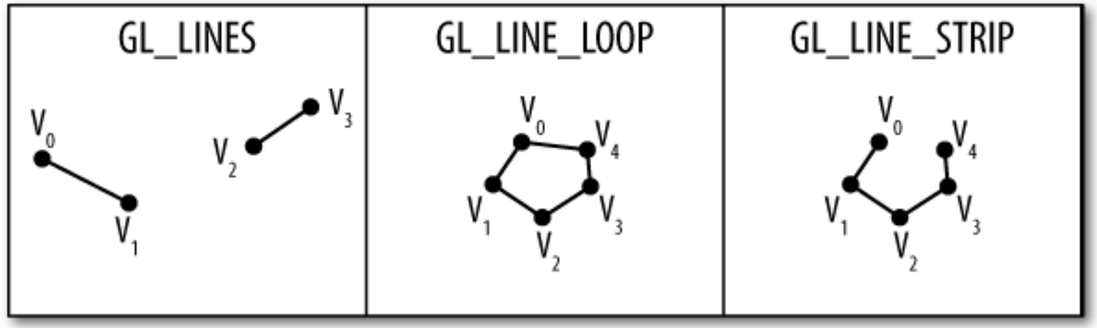


Vertices positioned on screen

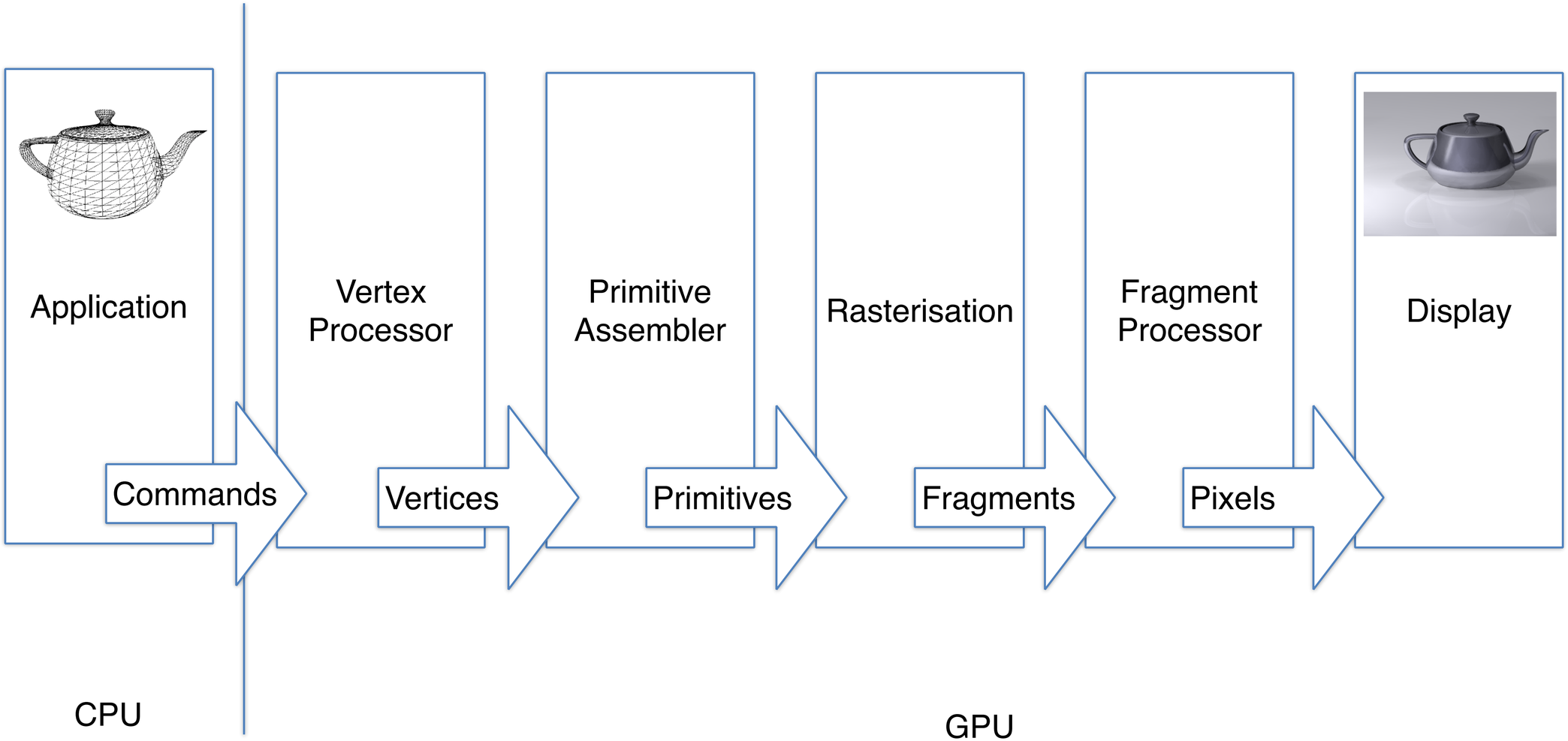
Primitive Generation → Primitive Processing → Face Culling



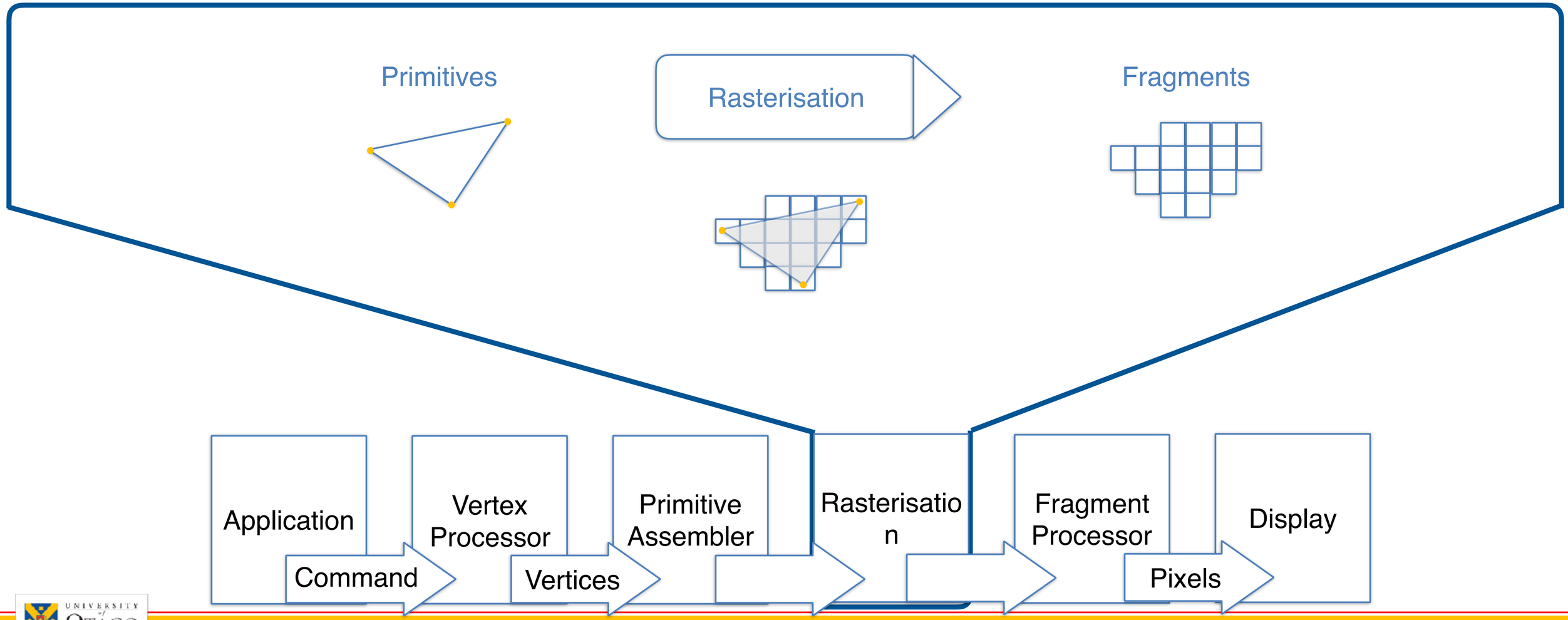
Primitives on screen



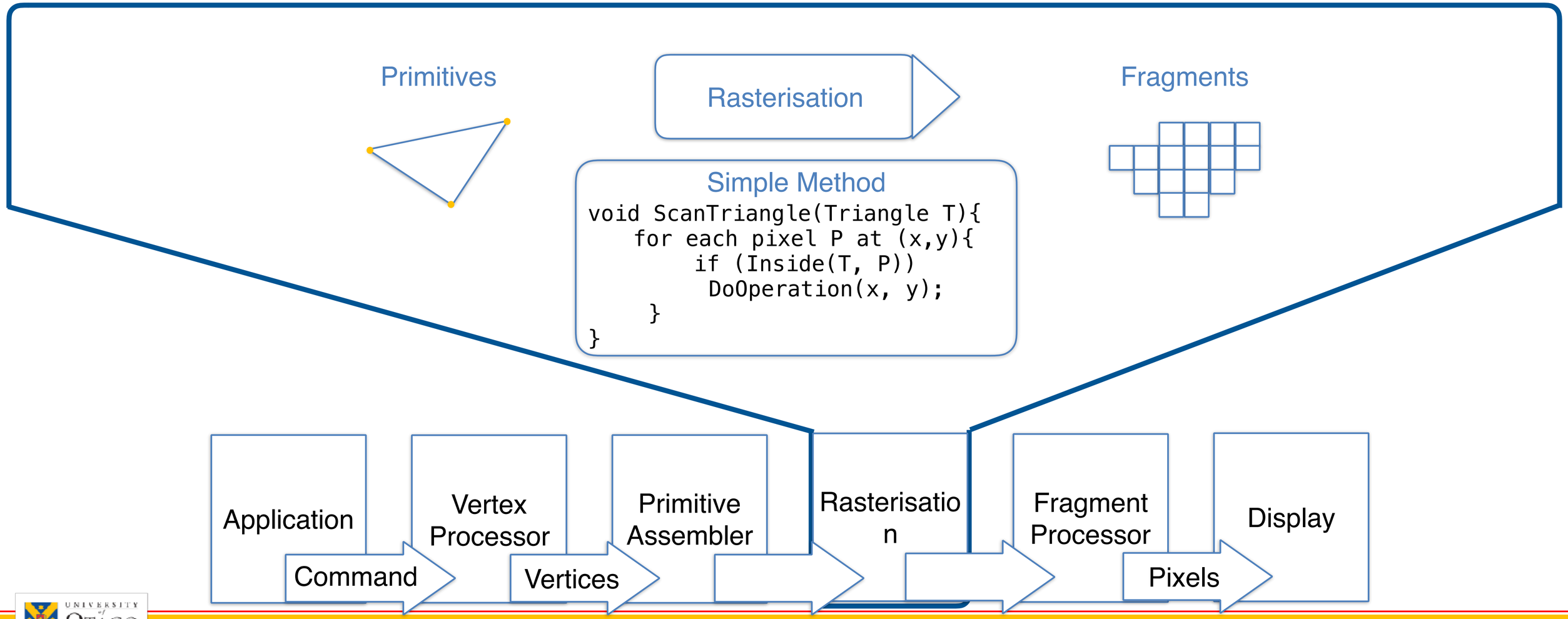
GRAPHICS PIPELINE



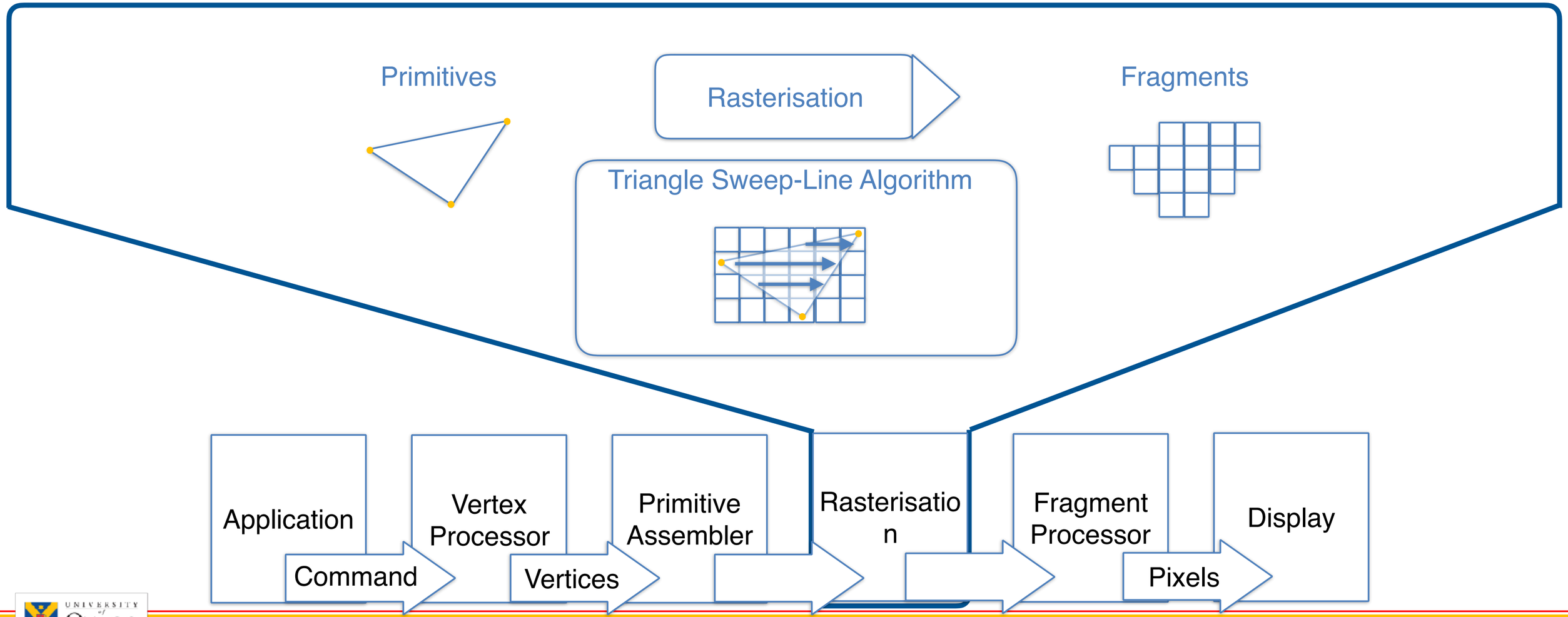
RASTERISATION



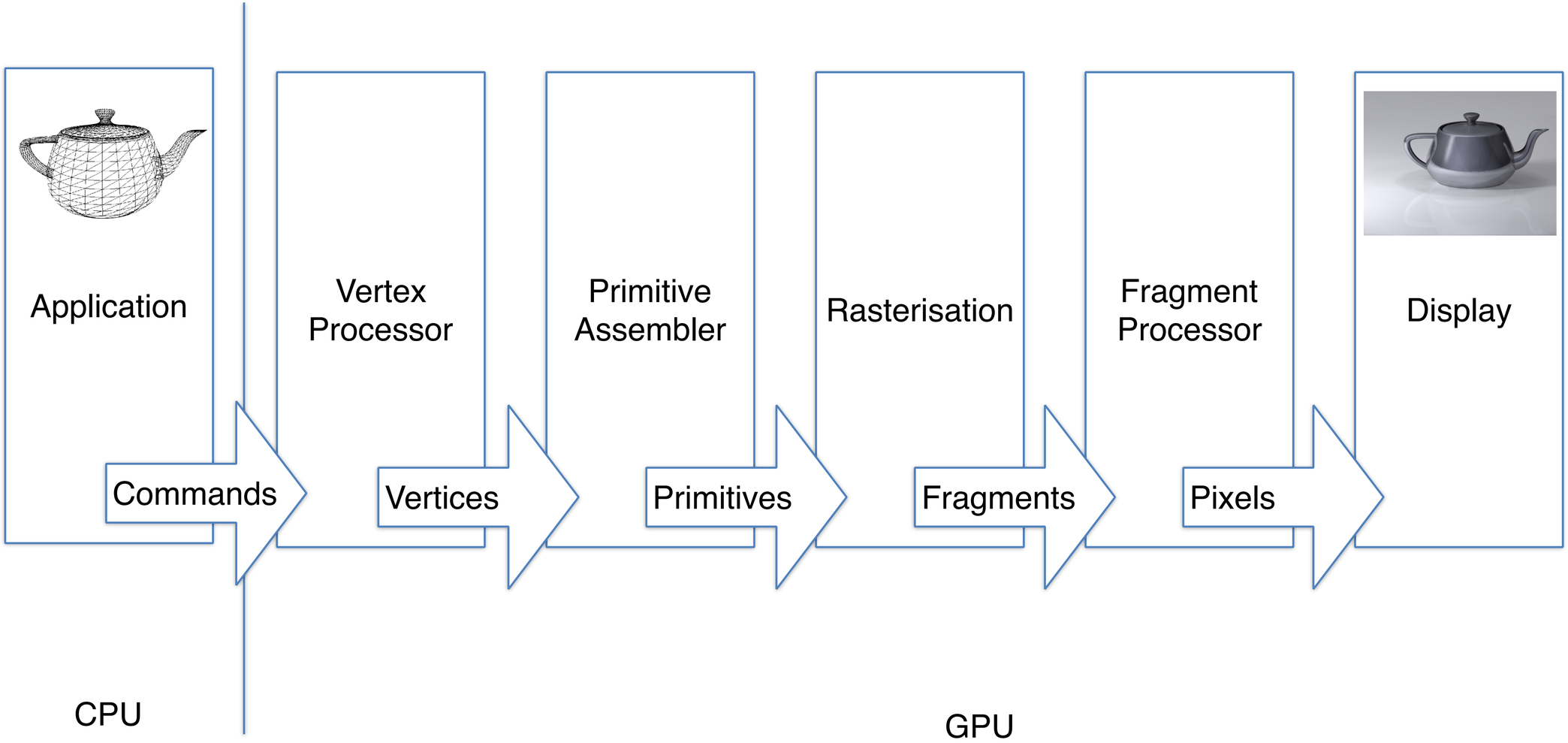
RASTERISATION



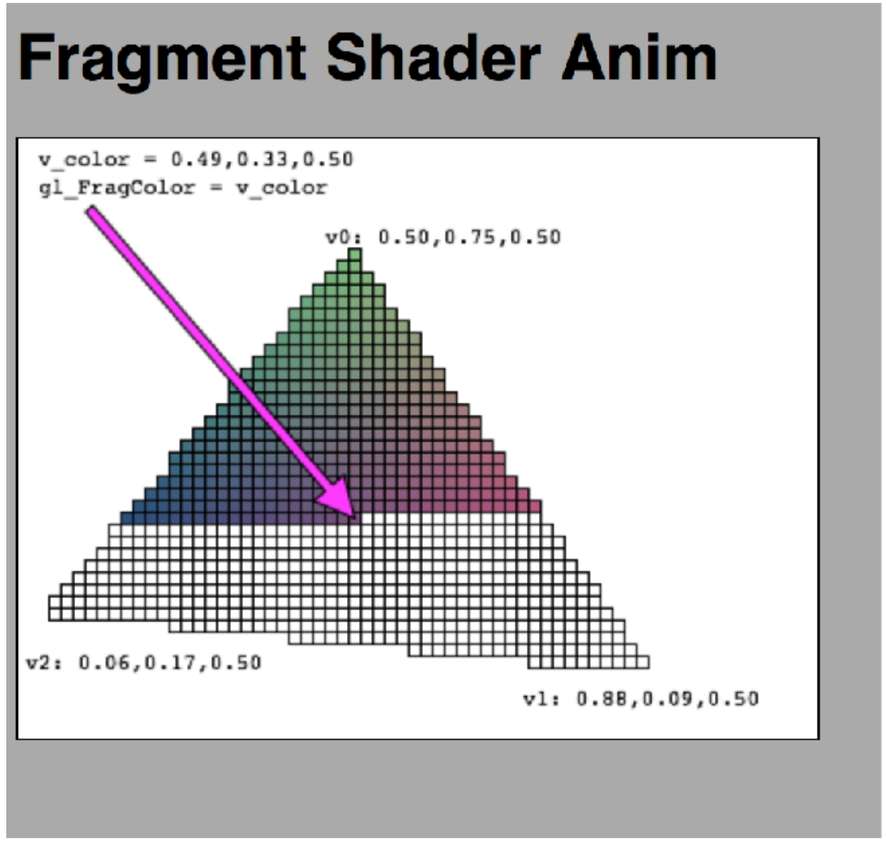
RASTERISATION



GRAPHICS PIPELINE



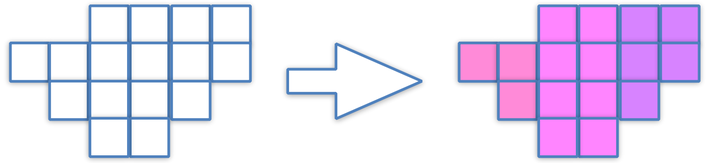
FRAGMENT PROCESSING



https://webglfundamentals.org/webgl/lessons/resources/fragment-shader-anim.html

FRAGMENT PROCESSING

- Output of the rasterisation stage are fragments
- Fragments are processed by a fragment shader
- Fragment shader have control over the color and depth values



FRAGMENT SHADER: EXAMPLE

```
// Ouput data
out vec3 color;

uniform vec4 colorValue;
void main()
{

  // Output color
  color = colorValue.rgb;

}
```

Thank You!

For more material visit

http://www.cs.otago.ac.nz/cosc342/