COMP30019 – Graphics and Interaction

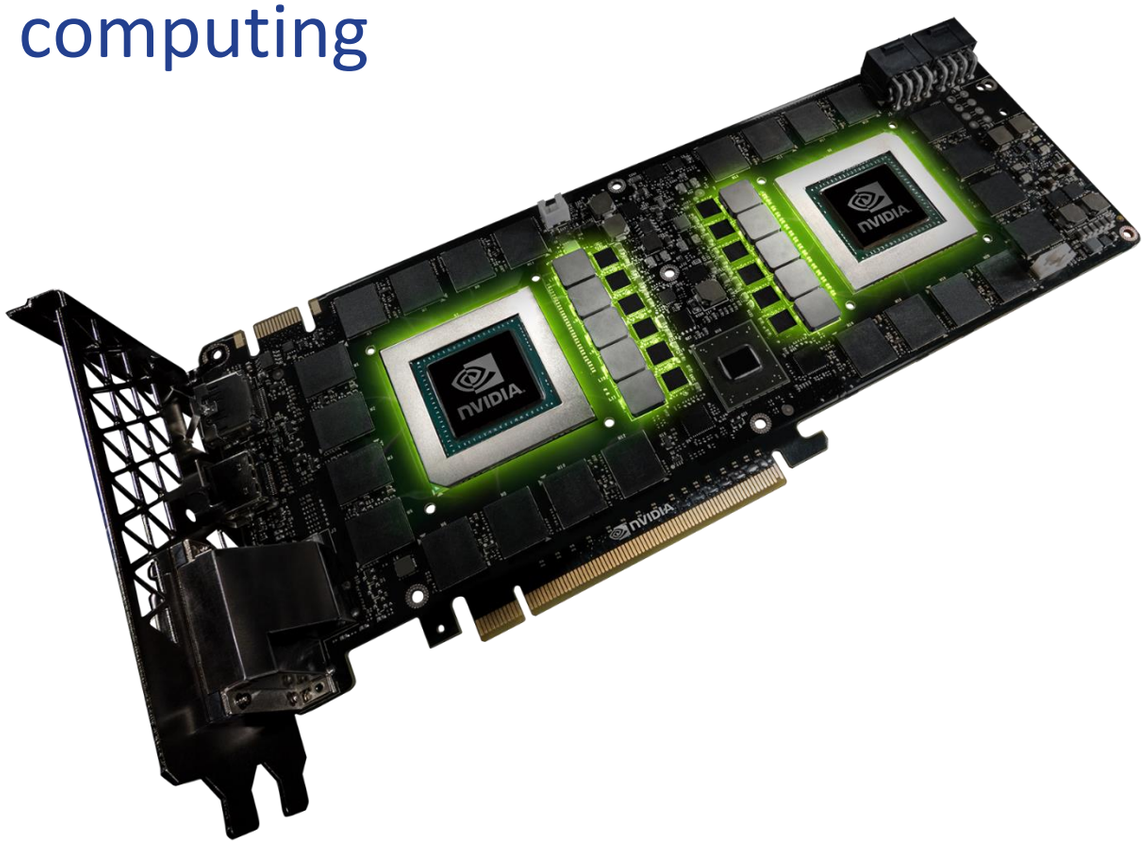
Lab 3:
Introduction to Shaders

Lecturer: Dr. Jorge Goncalves
Tutor:

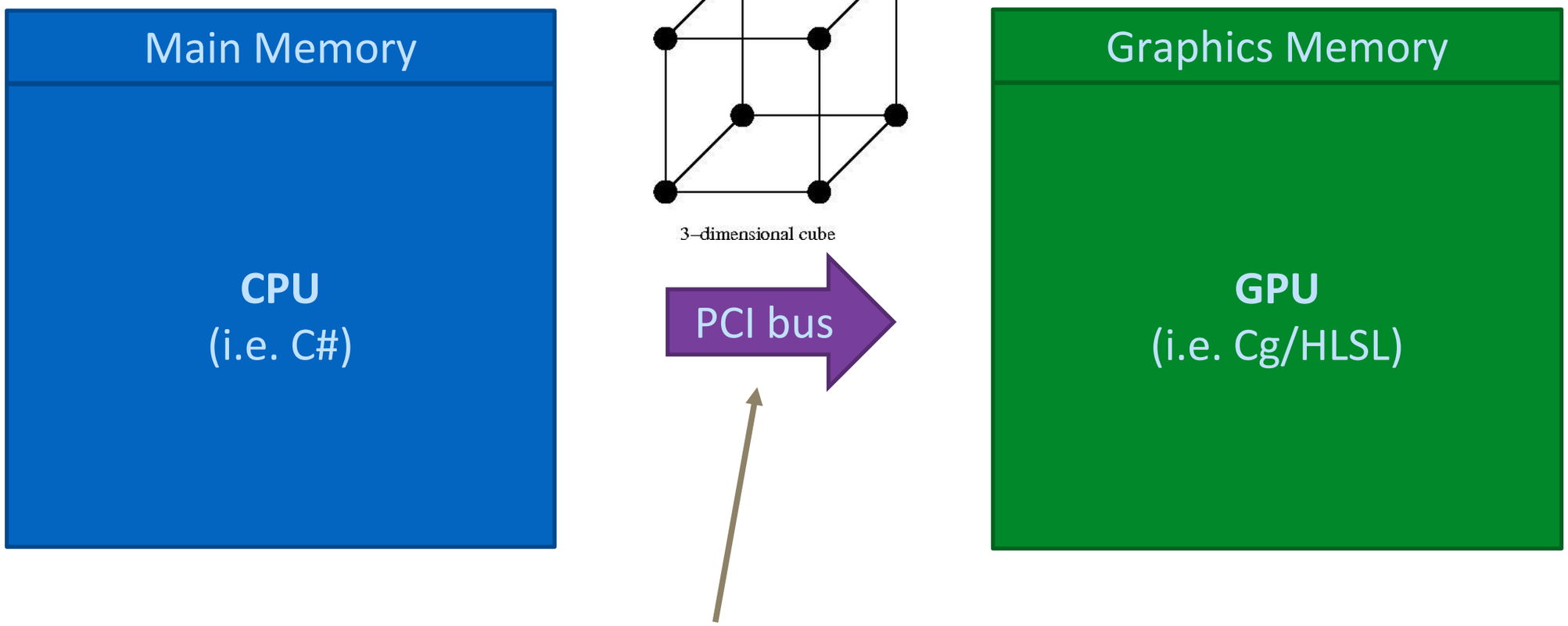
THE UNIVERSITY OF MELBOURNE

The GPU

- Located on a “graphics card”
- Specialised processor for graphics processing
- Strong emphasis on parallel computing



The GPU/CPU split



Upload mesh data structures *on occasion.* Limited Bandwidth!

Transformations

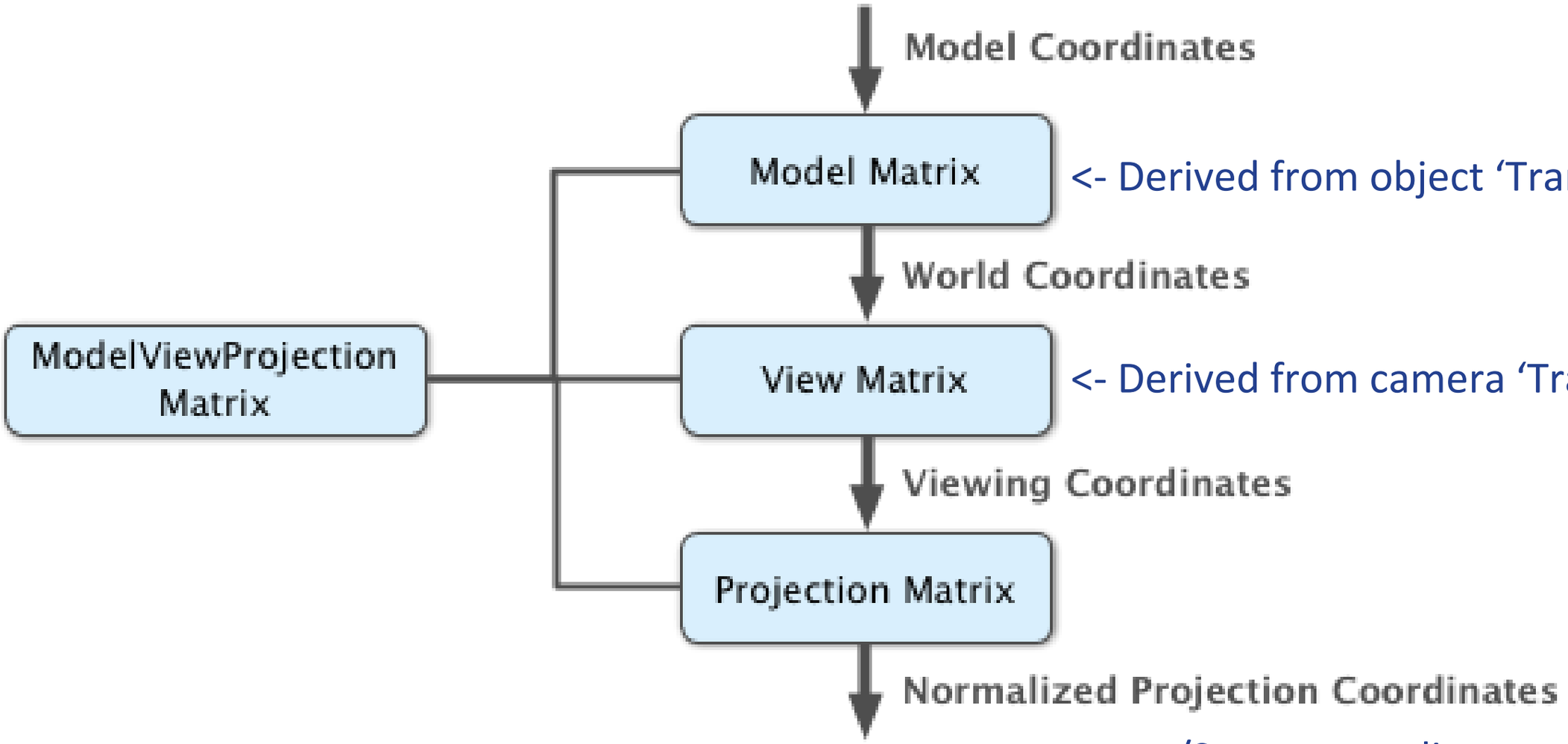
- If we only upload the vertices to the GPU once, then how to we **rotate, scale** and **translate** the object *in the game world* in real-time?

The Model Matrix

- Upload the vertices once, *then* upload a transformation matrix continually. **Let the GPU do the heavy lifting** of transforming all the vertices in real time via a shader.
- Unity automatically (behind the scenes) calculates this matrix based on the transform component of the object, and uploads it every frame!

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & 0 \\ m_{21} & m_{22} & m_{23} & 0 \\ m_{31} & m_{32} & m_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Model -> View -> Projection (MVP)



Credit: https://lva.cg.tuwien.ac.at

Programmability of the GPU

- In the past, the GPU had very limited programmability (i.e. everything was hardcoded into the hardware)
- Nowadays we can write programs called *shaders* which execute **on the GPU**.
- Programmability is ever increasing…

The Rendering Pipeline (Simplified)

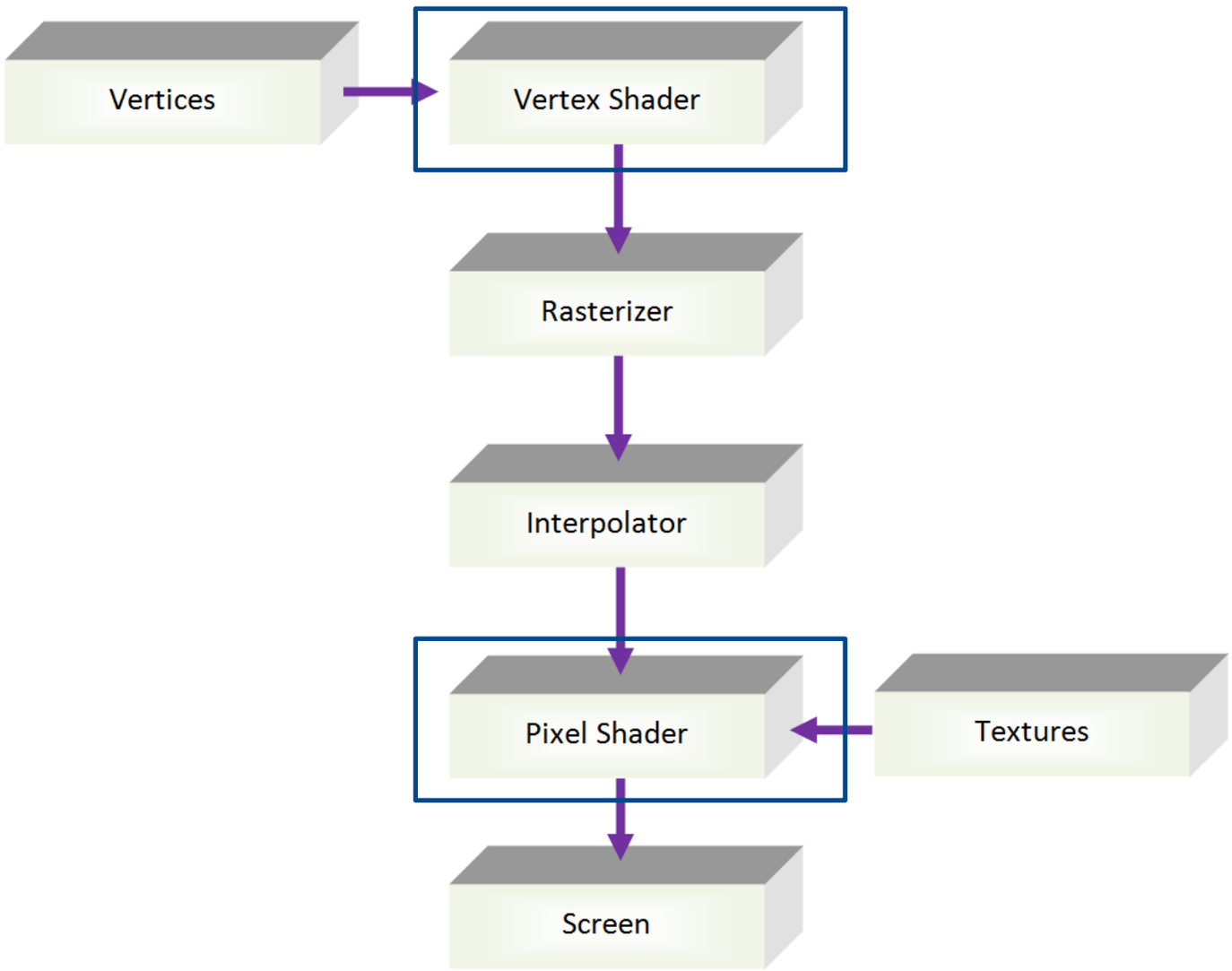


IMAGE CREDIT: http://centurion2.com/ComputerGraphics/CG150/CG150.php

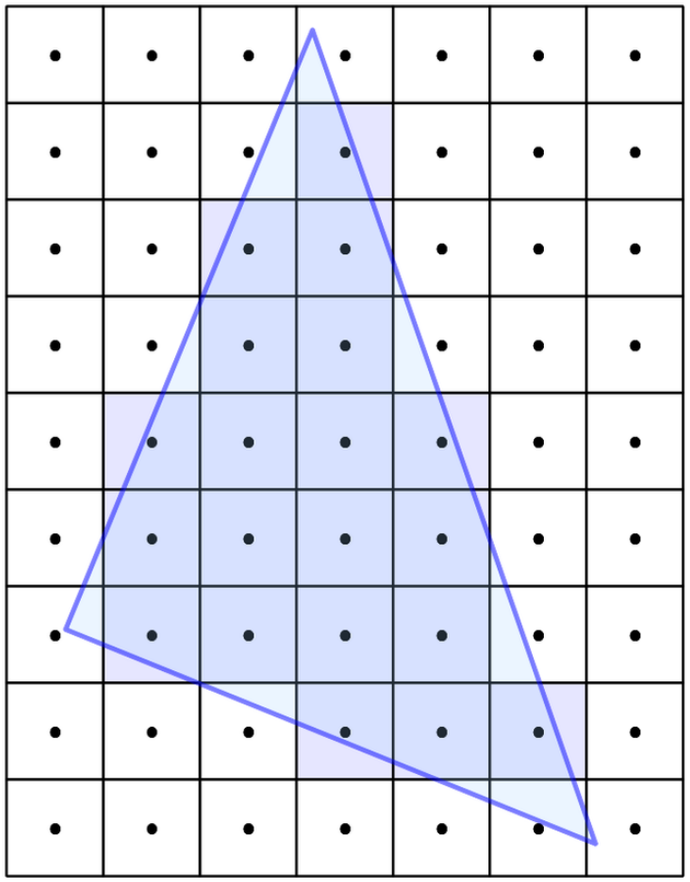
Vertex Shader

Vertex Shader

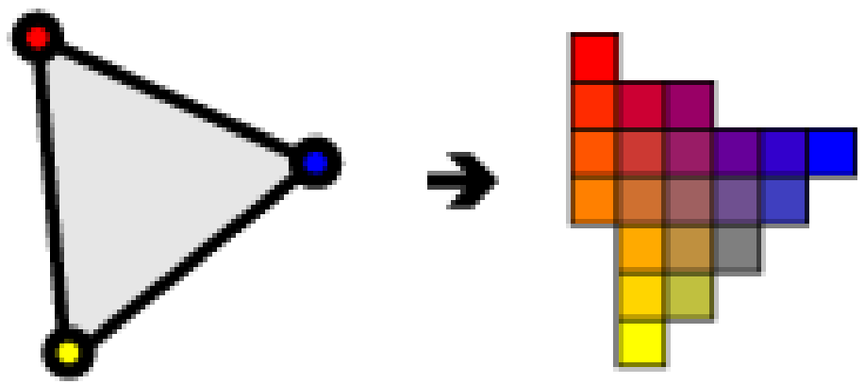
```
13          struct vertIn
14          {
15              float4 vertex : POSITION;
16          };
17
18          struct vertOut
19          {
20              float4 vertex : SV_POSITION;
21          };
22
23          // Implementation of the vertex shader
24          vertOut vert(vertIn v)
25          {
26              vertOut o;
27              o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
28              return o;
29          }
```

Vertex Shader -> Rasterizer

Rasterizer



https://en.wikibooks.org/wiki/GLSL_Programming/Rasterization



http://www.geometrian.com/

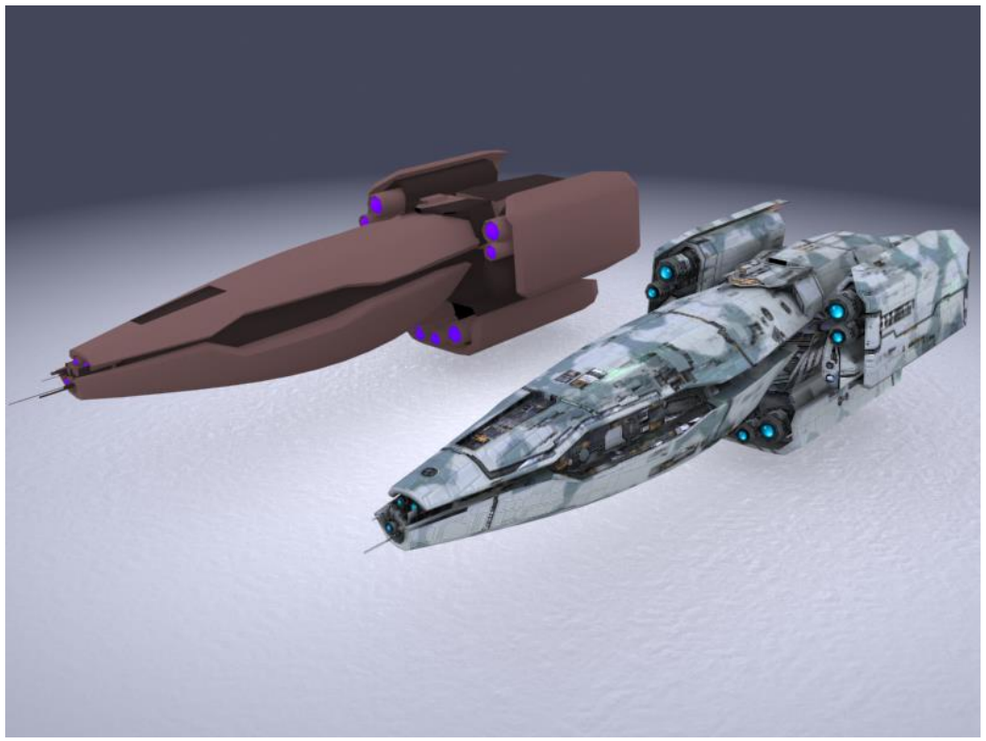
Fragment/Pixel Shader

Fragment/Pixel Shader

```
31          // Implementation of the fragment shader
32          fixed4 frag(vertOut v) : SV_Target
33          {
34              return float4(0.0f, 0.0f, 0.0f, 1.0f);
35          }
```

Textures

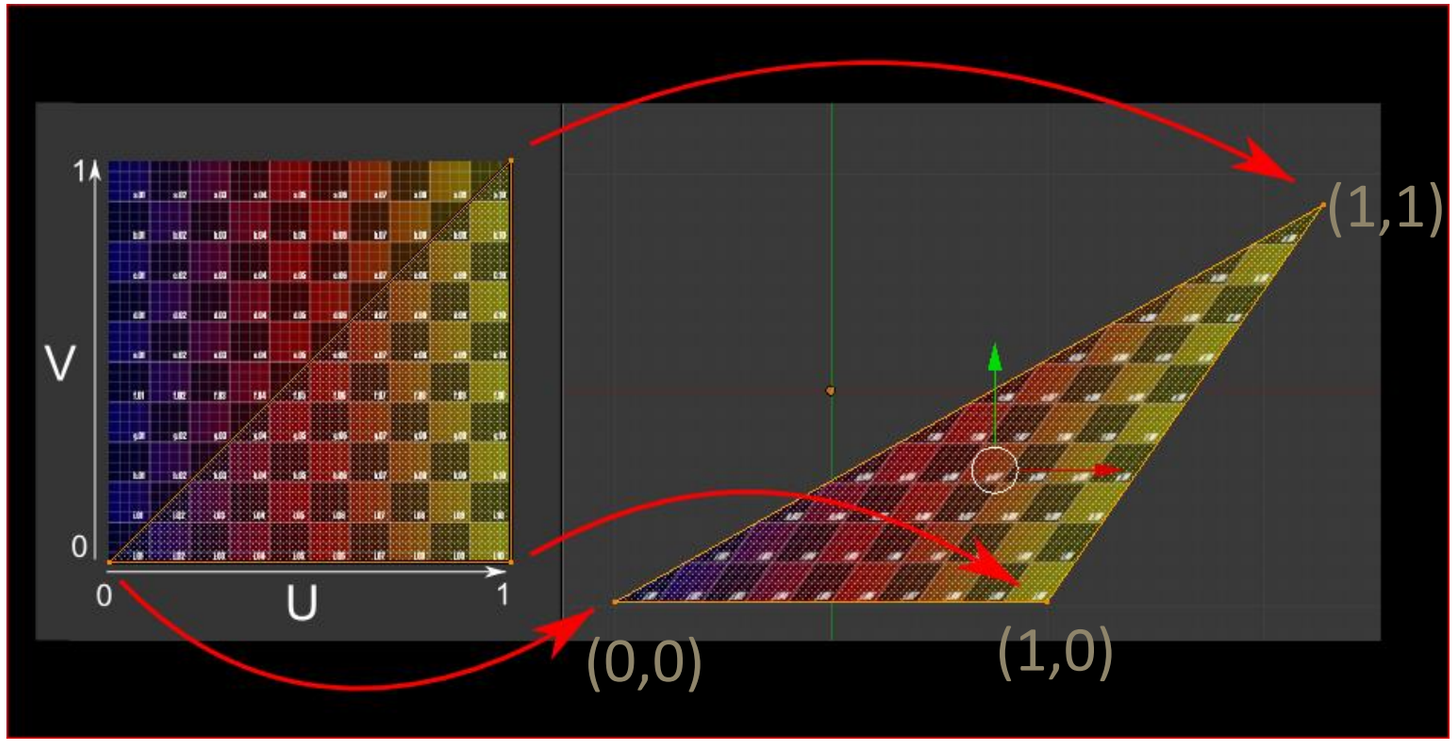
- Vertex colours are *limiting*
- A texture is an image that is wrapped onto the surface of a mesh



SOURCE: http://www.indiedb.com/games/angels-fall-first-the-second-antarean-war/images/fox-before-and-after-texturing

Textures

- Idea: Associate a *UV coordinate* with each vertex (like with vertex colours).



Textures

- Textures are implemented in shaders by means of a *texture sampler*.
- The interface to a texture sampler in HLSL is a **function**.
  - Inputs: **Texture** and a **UV coordinate**
  - Output: A **colour**
- You will explore this in the latter part of today's lab.