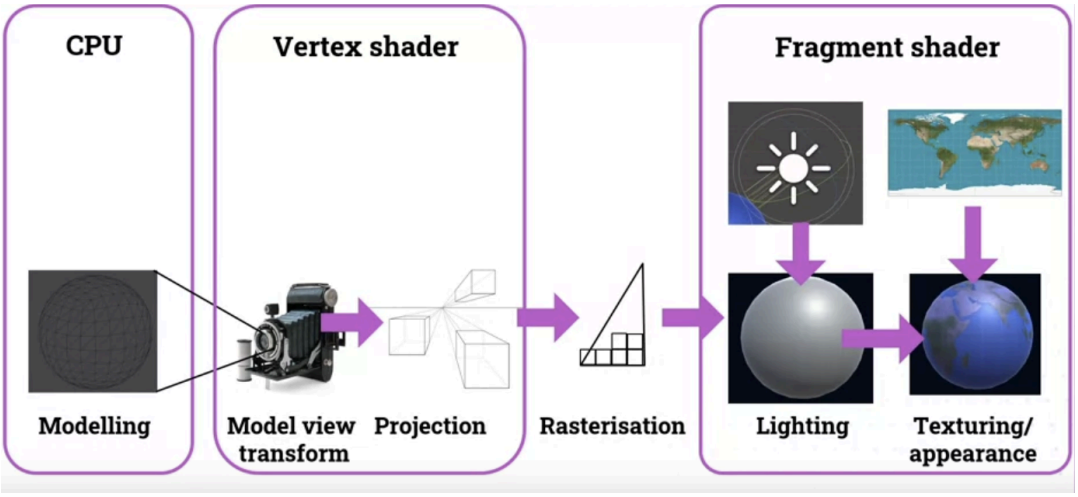
Topic 7 - GPU Programming

Graphics hardware and graphics processing units, also called GPUs, what have enabled the amazing quality of graphics we can see now in modern films and games

Vertex Shader: This is a small program that runs on the GPU that processes individual vertexes, after that the vertexes are combined into polygons and rasterized.



Vertex shader happens to vertexes and fragment shaders happen to fragments, those little patches of an object that will become pixels.

CPU Processor

CPU is a **serial** processor, what does that mean? It means that it performs instructions one at a time, in particular, if you're processing polygons or vertexes, it will process each polygon or vertex one at a time.

GPU Processor

GPU is a parallel processor so that multiple instructions can be happening at the same time. And that means we can process multiple vertexes, multiple fragments simultaneously, massively speeding up the process of rendering.

Why is the GPU so different that it can be so parallel in comparison to a CPU?

- Each vertex that you're processing is **independent of the other vertexes**, so there's no dependencies or data sharing between vertexes or fragments.
- So the only data that's shared between them doesn't change, it's the same across all vertexes

- It **doesn't matter what order you do them** in and in particular means you can do them in any order and at the same time in parallel.
- Dedicated Language GLSL, HLSL
- The open GL graphics standard has a language called GLSL for the GL shader language, and Microsoft developed the high level shader language, HLSL, which is what is used by Unity.

Q What is the relationship between GPUs and programming languages

The GPU and CPU code are in different programmes and GPUs use different languages from CPU programmes

Q: Think of an example of something that might be improved in a graphics scene by having a better GPU, and something that will not.

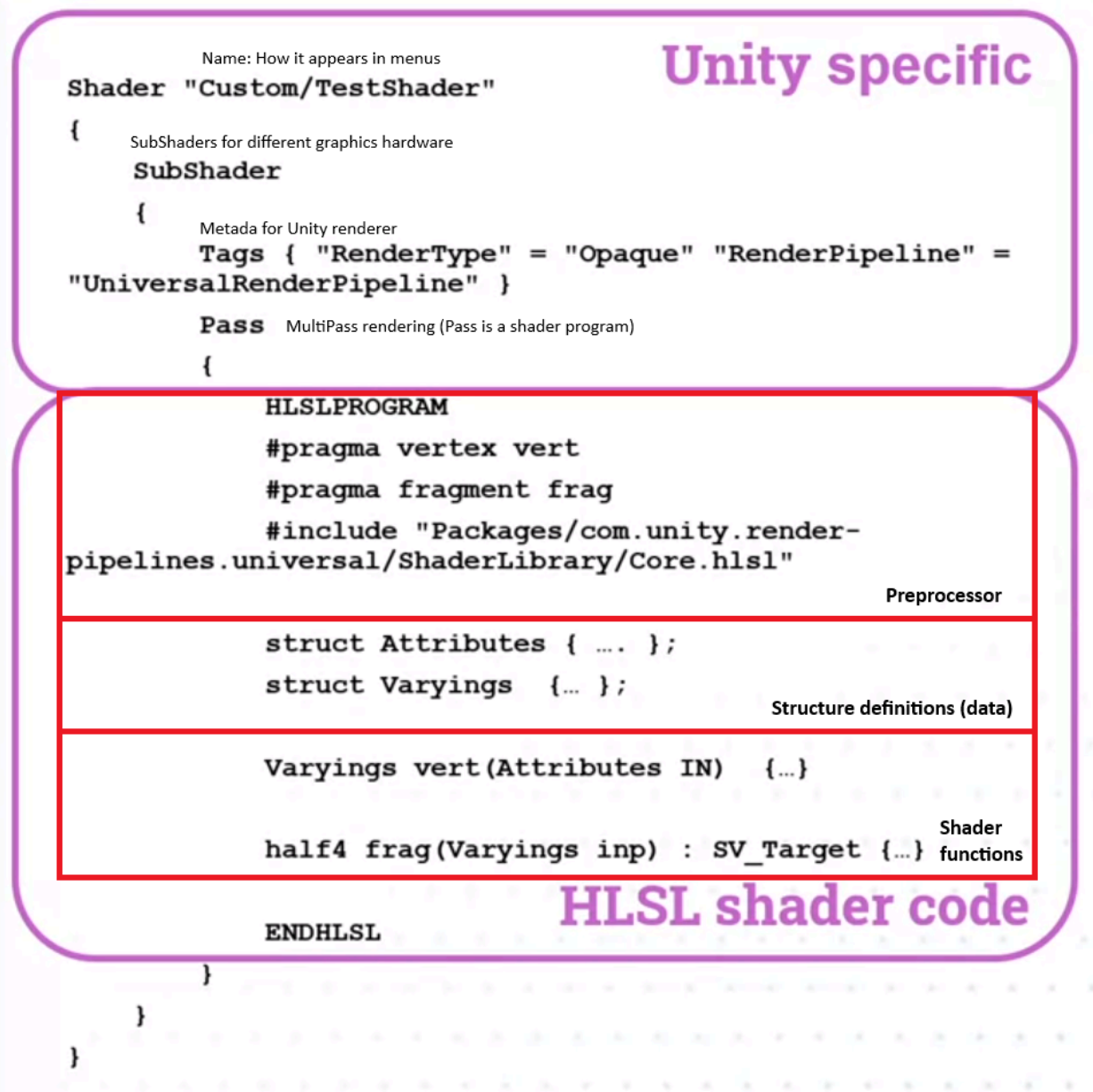
Having a better GPU will almost always increase the framerate. With a better GPU you can also have more polygons, higher resolution textures, better lighting effects and more complex surface properties.

The GPU won't have much effect on things like animation, physics or AI.

How Vertex Shaders Work

- First part of the GPU pipeline
- They Transform and project vertices
- Vertex Shaders **Inputs:** if you're doing things like modeling, transforms, and projections, you need those **transform matrices**, **modelview matrix**, **projection matrix**
- These inputs don't change between vertices; they're the same for all vertices and objects. So we call these **uniform variables**.
- Other types of data like **vertex position,** which is different for each vertex, but is specific only to that vertex so that you don't share the vertex position between vertices and we call this a **vertex attribute.**
- The Vertex Shader provides **output** that is **rasterized** and used by **fragment shaders.** This output is called the **Varying variable** (EG Transformed vertex position)
- Varyings are **interpolated** in the rasterization face for the Fragment Shaders, .

Shader Program



Vertex Shader Example

```
Varyngs vert(Attributes IN)
{
    Varyngs OUT;
    OUT.pos = mul(UNITY_MATRIX_M, IN.pos);  // model view transform
    OUT.pos = mul(UNITY_MATRIX_V, OUT.pos); // model view transform
    OUT.pos = mul(UNITY_MATRIX_P, OUT.pos); // Projection
    return OUT;
}
```

UNITY_MATRIX variables are the **Uniform** variables

Shortcut for doing the same as avobe

```
Varyngs vert(Attributes IN)
{
    Varyngs OUT;
    OUT.pos = mul(UNITY_MATRIX_MVP, IN.pos);
    return OUT;
}
```

The one you should use by Unity

```
Varyngs vert(Attributes IN)
{
    Varyngs OUT;
    OUT.pos = TransformObjectToHClip(IN.pos);
    return OUT;
}
```

An example of creating an effect

```
Varyngs vert(Attributes IN)
{
    Varyngs OUT;
    OUT.pos = IN.pos;
    OUT.pos = 0.5sin(IN.pos.y) + 1.0;
    OUT.pos = TransformObjectToHClip(OUT.pos);
    return OUT;
}
```