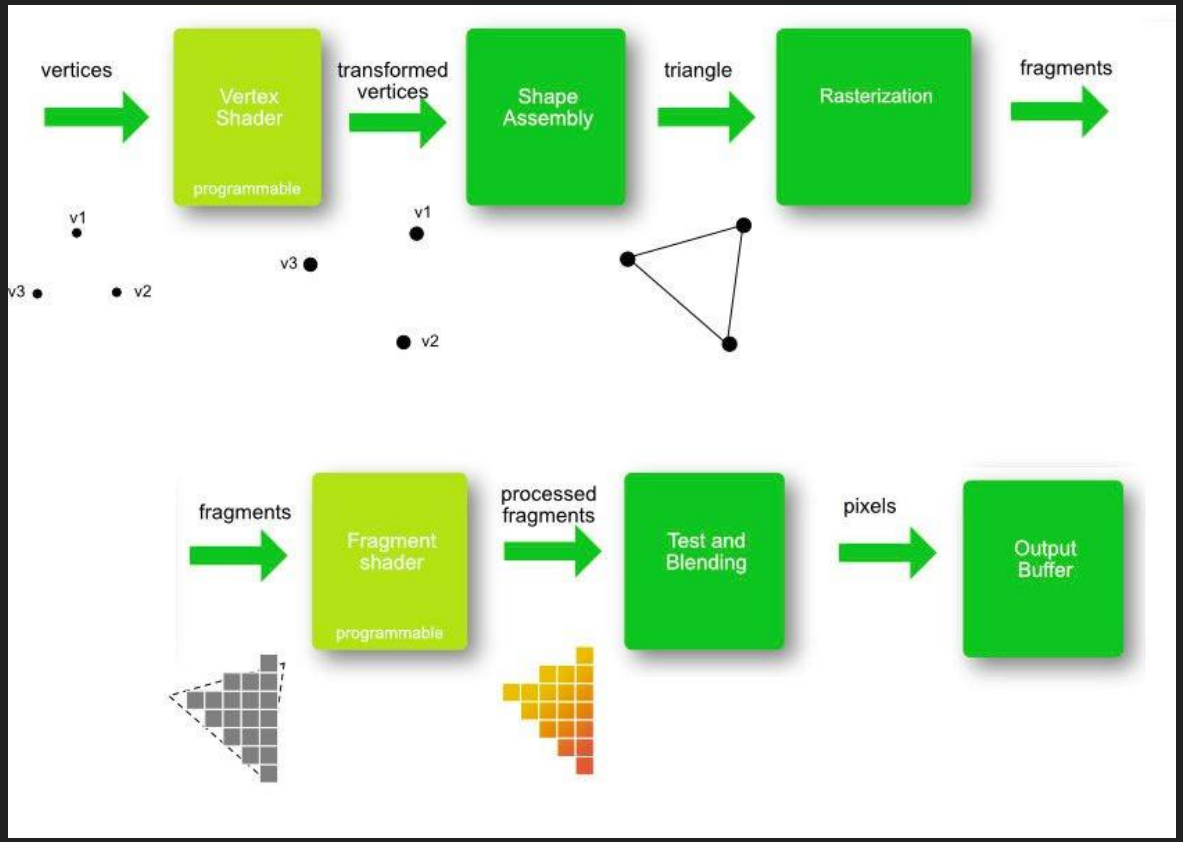
Intro to Unity Shaders

CM163 Lab 1

Rendering Pipeline

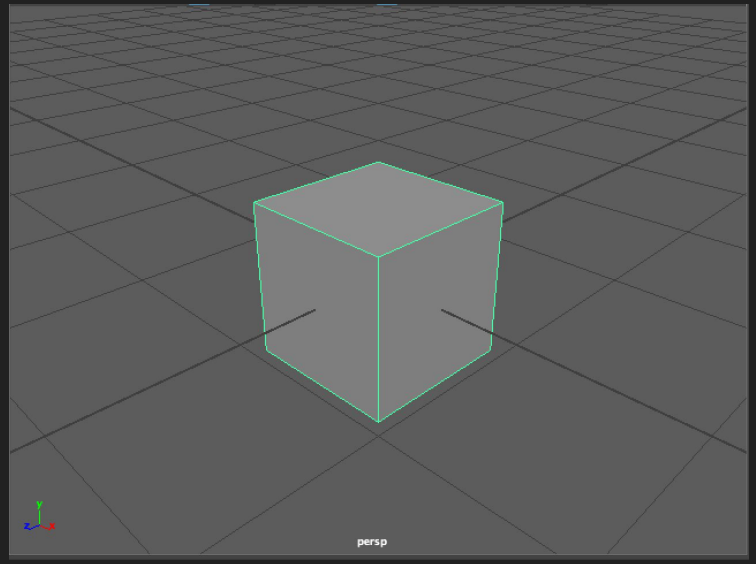


Vertex Shader - Program that transforms vertices in someway

Rasterizer - Turn the transformed vertices to pixels on the screen

Fragment Shader - Program that process the pixels

Creating a 3D model



Object Space

Position of vertices are defined wrt to the center of this coordinate space

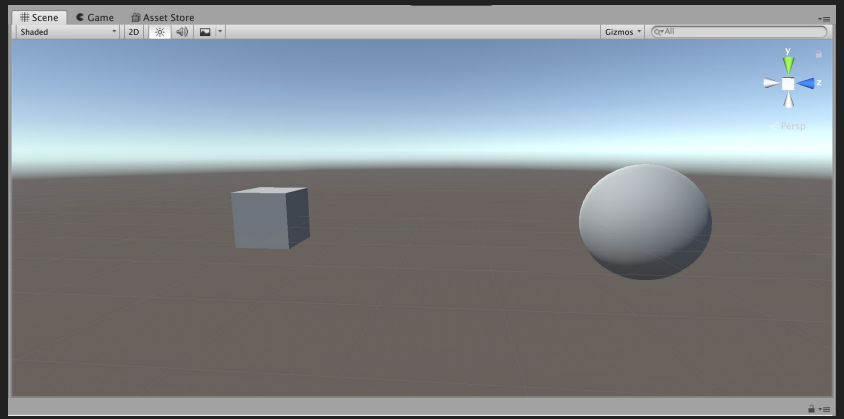
cube.obj

```
# This file uses centimeters as units for non-parametric coordin

mtllib cube.mtl
g default
v -0.500000 -0.500000 0.500000
v 0.500000 -0.500000 0.500000
v -0.500000 0.500000 0.500000
v 0.500000 0.500000 0.500000
v -0.500000 0.500000 -0.500000
v 0.500000 0.500000 -0.500000
v -0.500000 -0.500000 -0.500000
v 0.500000 -0.500000 -0.500000
vt 0.375000 0.000000
vt 0.625000 0.000000
vt 0.375000 0.250000
```

Using any 3D modeling software such as blender or Maya

Importing to Unity



World Space

Position of 3D objects are defined wrt to this coordinate space

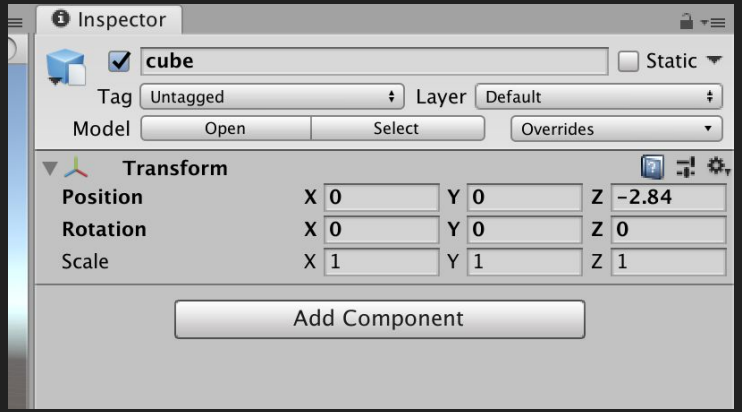
How to transform vertices from object space to world space?

Model Matrix

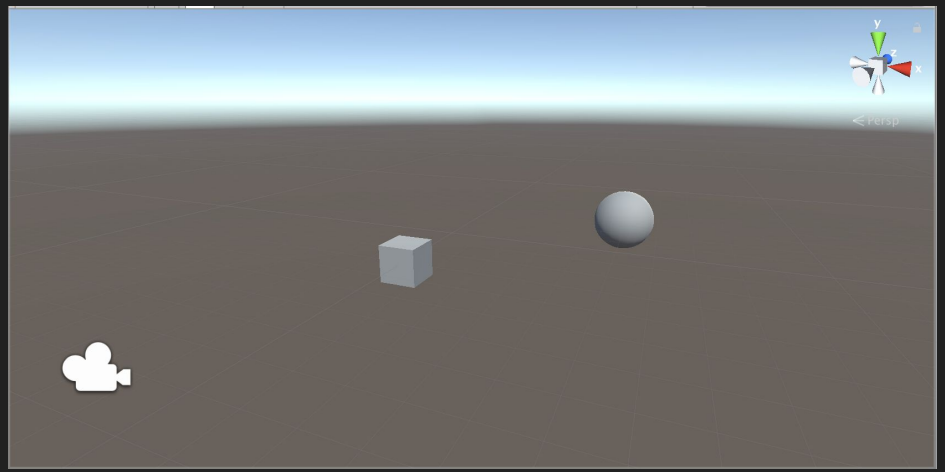
Transform vertices from object space to world space

It’s a 4x4 matrix which is defined by Unity when we load a 3D mesh

Performs translation, rotation and scaling in world space.



Camera space



View Space

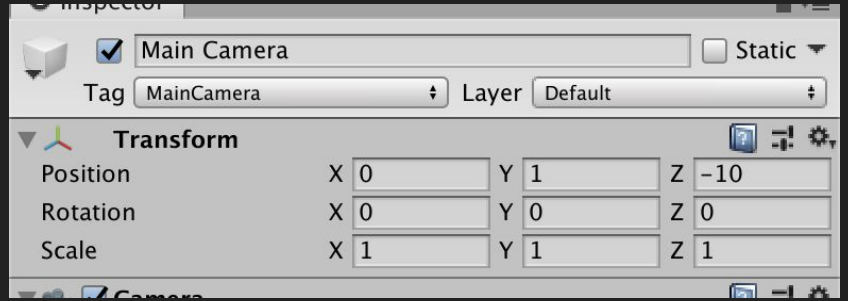
Position of 3D objects are defined wrt to camera coordinate system

View Matrix

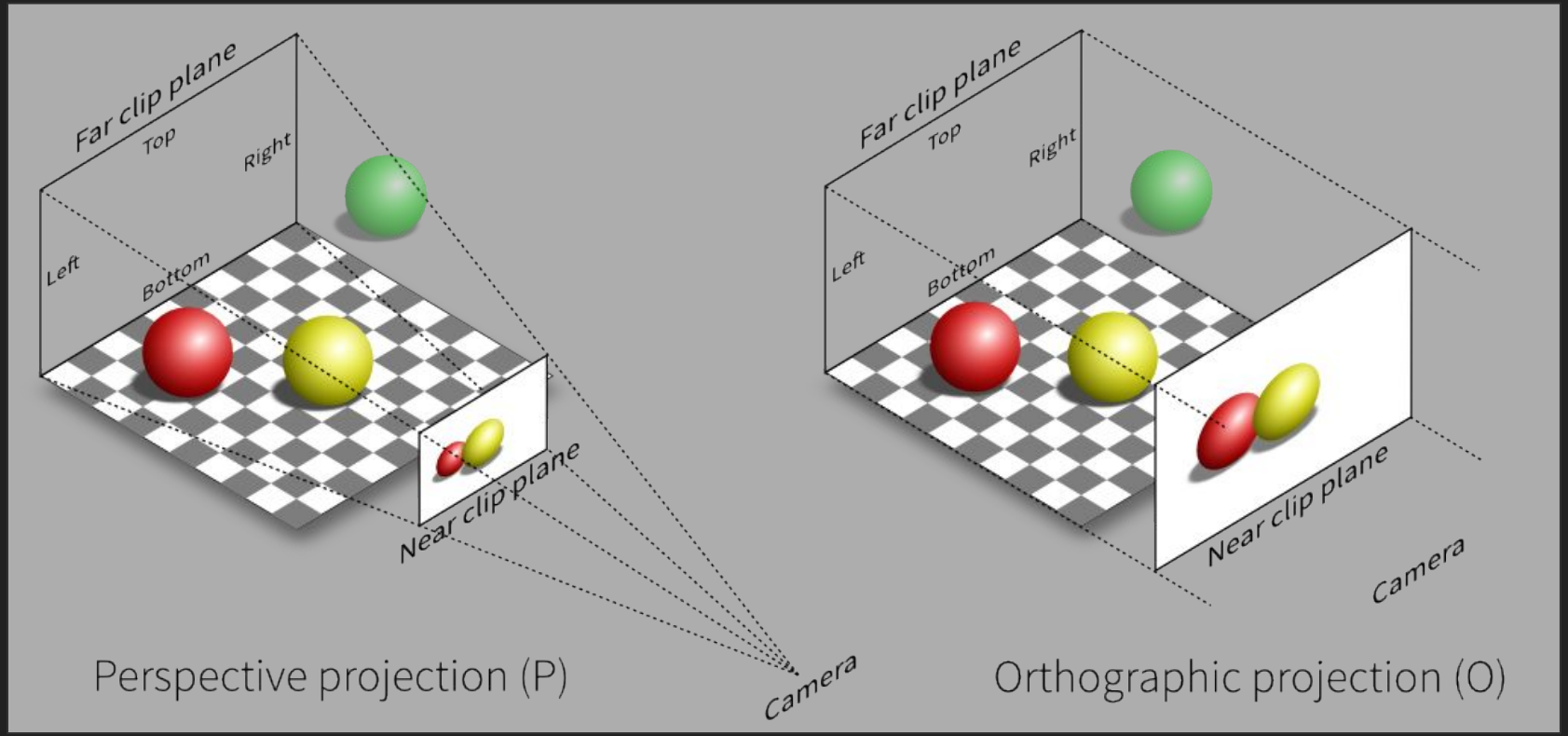
Transform vertices from world space to camera space

It’s a 4x4 matrix which is defined by Unity when we create a Camera

Performs translation, rotation and scaling in view space.



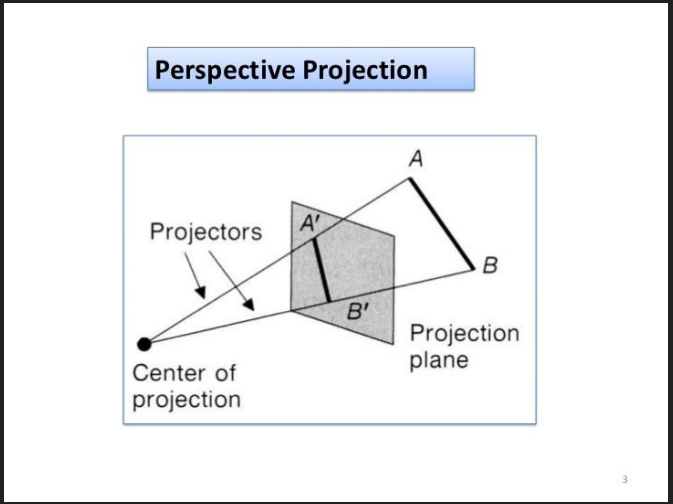
Projection Space

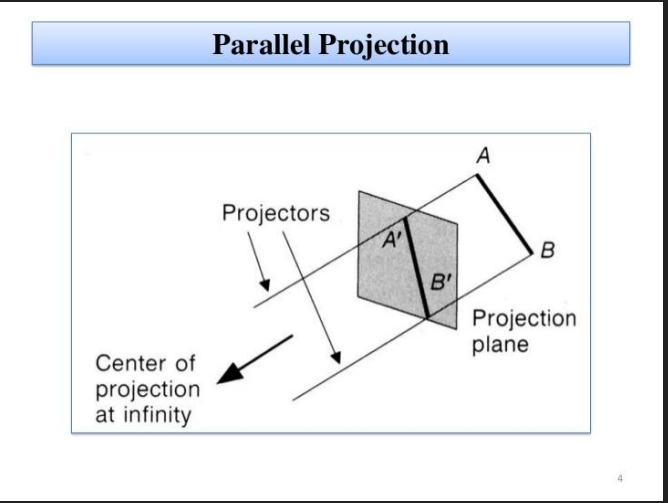


Projection Matrix

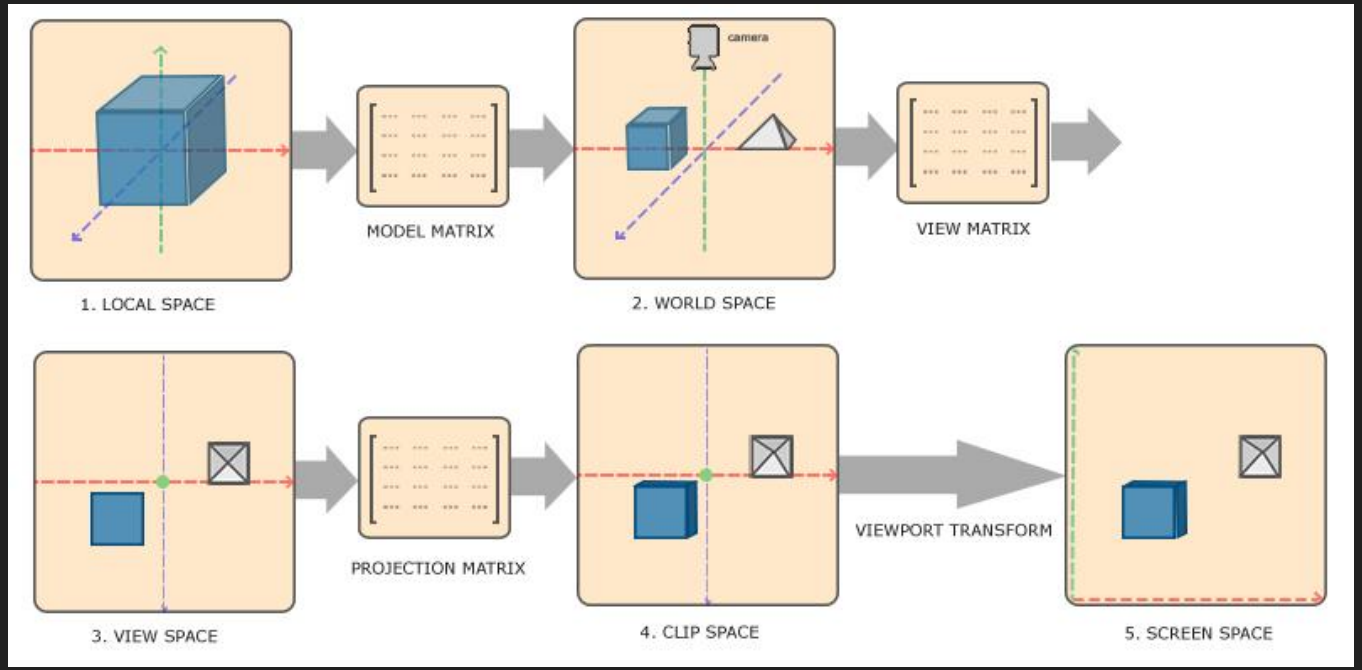
Transform vertices from camera space to a 2D space

It’s a 4x4 matrix which is defined by Unity when we create a Camera





MVP (Model View Projection) Matrix



Vertex * Model matrix * View Matrix * Projection Matrix

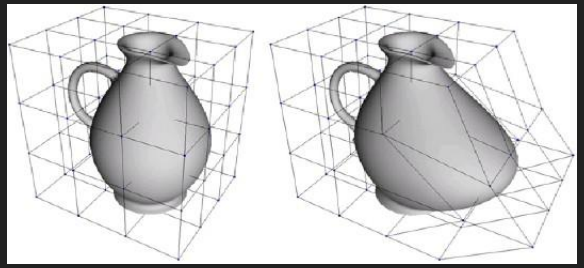
Vertex Shader

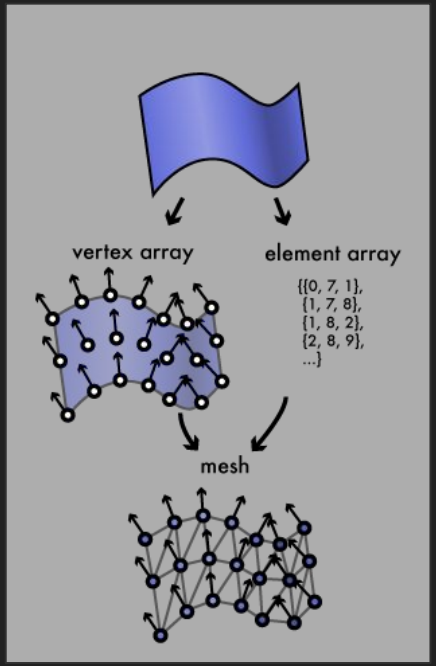
Program that transforms vertices in someway

Performs MVP operation

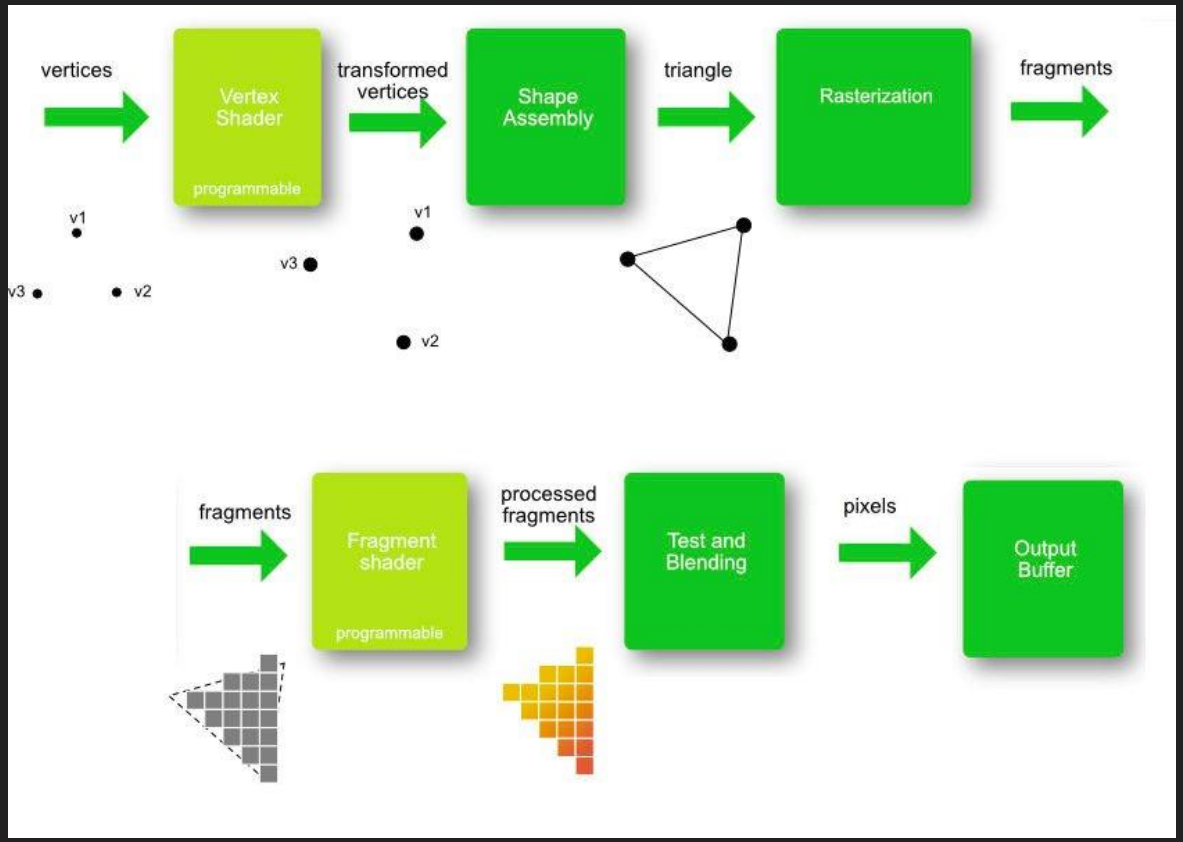
Other uses for vertex shaders:

- Object deformation
- Vertex animation
- Water ripples
- Sending values to pixel shader
  - Position
  - Normal
  - Color





Rendering Pipeline



Vertex Shader - Program that transforms vertices in someway

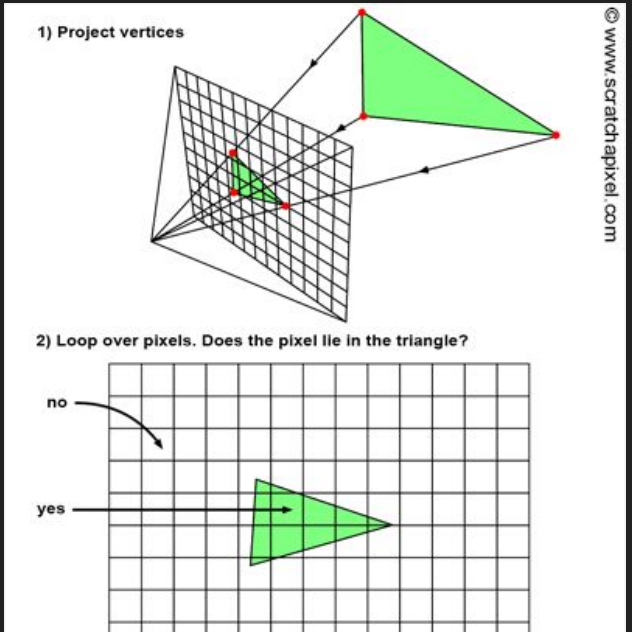
Rasterizer - Turn the transformed vertices to pixels on the screen

Fragment Shader - Program that process the pixels

Rasterization

Fixed function - not programmable

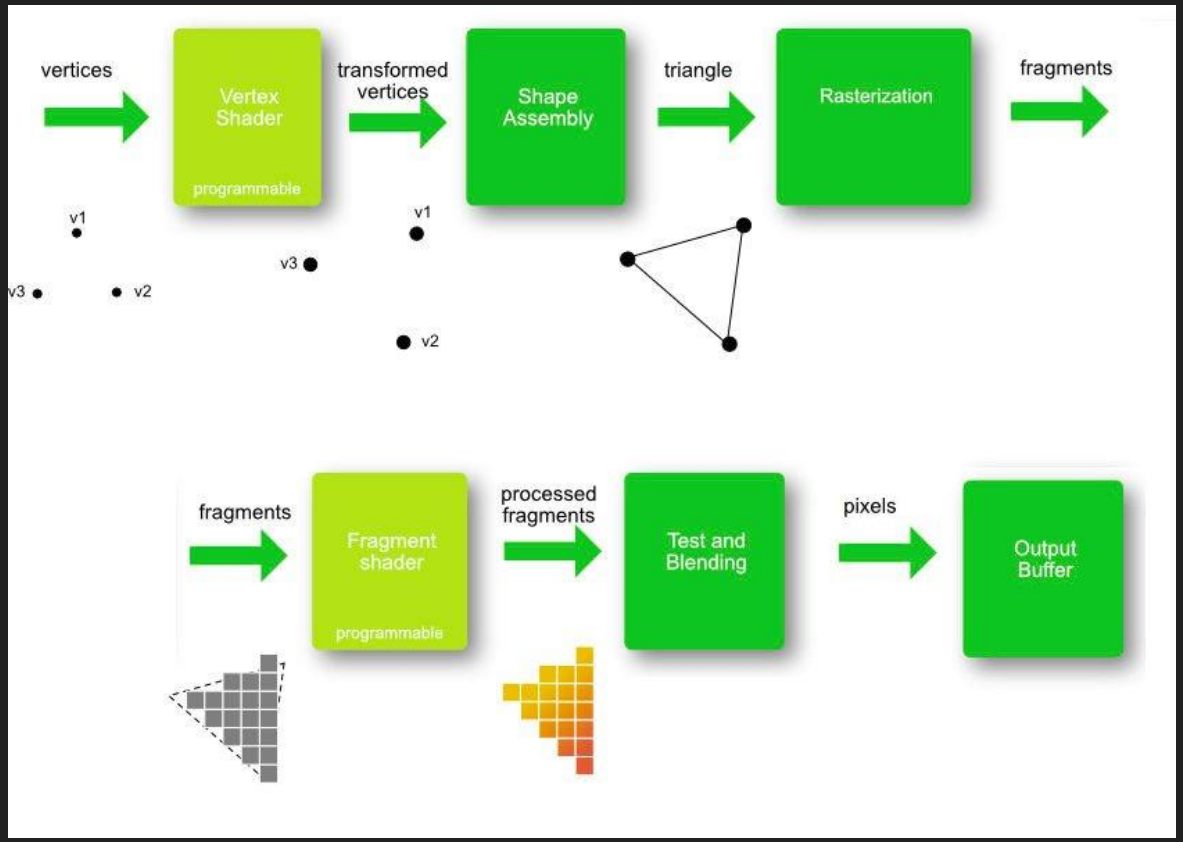
The main function of a rasterizer is to find the pixels on the screen that is covered by the triangles



It also interpolates the values sent by vertex shader:

- Position
- Normal
- Color
- Etc

Rendering Pipeline



Vertex Shader - Program that transforms vertices in someway

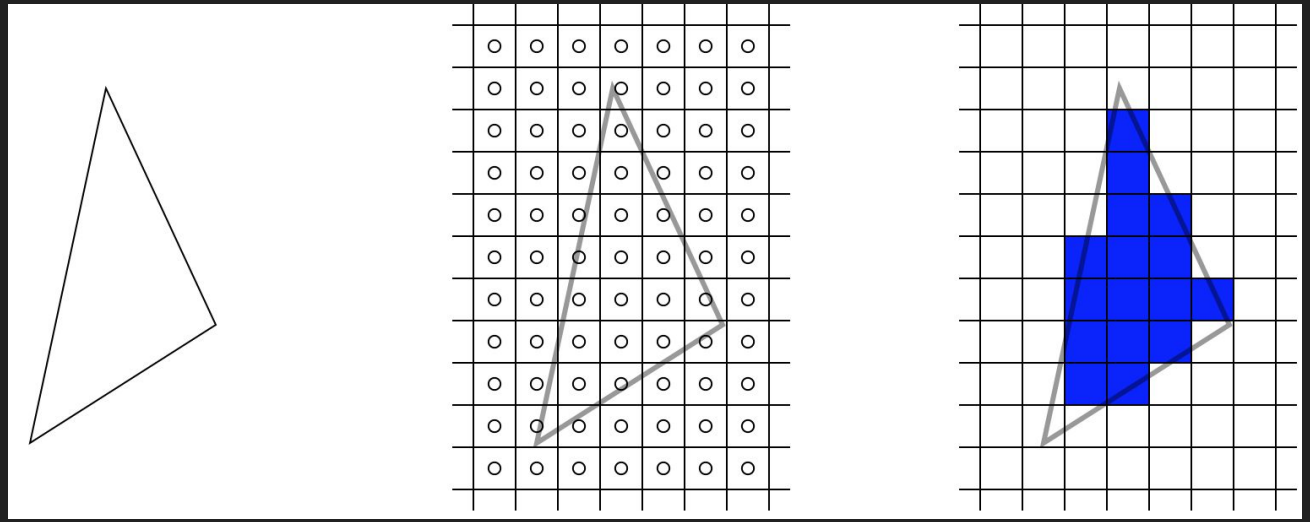
Rasterizer - Turn the transformed vertices to pixels on the screen

Fragment Shader - Program that process the pixels

Fragment Shader

Program that process the pixels

Mainly used for light calculations and computing pixel colors



Unity Shader

ShaderLab + CG/HLSL

ShaderLab provides an interface between Unity and Shader code

CG/HLSL - C for graphics / High level shader language

Unity Shader from scratch

```
Shader "CM163/FirstShader"
{

}
```

// ShaderLab

Shader “directory/shader name”

Properties

```
Shader "CM163/FirstShader"
{
    Properties
    {
        _Color("Main Color"  ColPor) = (1 1 1 1)

      // Variable name (label, data type) = default value
    }

}
```

// Properties

Input for the shader set by the user in the material inspector

```
Properties
{
    _Int ("Integer", Int) = 1
    _Float ("Float", Float) = 1.0
    _Vector4 ("Vector4", Vector) = (1,1,1,1)
    _Color ("Color", Color) = (1,1,1,1)
    _Range ("Range", Range(0,1)) = 0.5
    _2DTex ("2d Tex", 2D) = "defaulttexture" {}
    _CubeTex ("Cube Tex", Cube) = "defaulttexture" {}
    _3DTex ("3d Tex", 3D) = "defaulttexture" {}
}
```

SubShader

```
Shader "CM163/FirstShader"
{
    Properties
    {
        _Color("My Custom Color"  Color) = (1 1 1 1)
    }

    SubShader
    {

    }

   SubShader
    {

    }

}
```

// Sub Shader

Unity shader can have different sub shaders to support different hardware features

For eg: one subshader for iPhone and another one for Playstation

Passes

```
Shader "CM163/FirstShader"
{
    Properties
    {
        _Color("My Custom Color"  Color) = (1 1 1 1)
    }

    SubShader
    {
        Pass
        {
        }
        Pass
        {
        }
    }

}
```

// Passes

Each SubShader can have multiple render passes

Each pass will have a vertex shader and a pixel shader

One pass is one drawcall

Depth Pass
Lighting Pass
Post-processing Pass

CG

```
Shader "CM163/FirstShader"
{
  ....
        Pass
        {
                CGPROGRAM


                ENDCG
        }

}
```

// CGPROGRAM

This is where we write our shader code

Defining vertex and fragment shader functions

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            ENDCG
        }

}
```

Pragma is a compiler directive

Vertex/Fragment is the command

Vert and Frag are the name of the functions

Getting data from Unity world in Shader world

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            struct VertexShaderInput
            {
                float4 vertex : POSITION;
            };

            ENDCG
        }

}
```

```
Struct “name”
{
        Position
        Normal
        Color
        TexCoords
        etc
}
```

Getting data from Vertex Shader in Frag Shader

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
                CGPROGRAM
                #pragma vertex vert
                #pragma fragment frag

                struct VertexShaderInput
                {
                    float4 vertex: POSITION;
                };

                struct VertexShaderOutput
                {
                    float4 pos: SV_POSITION;
                } ;

                ENDCG
        }
```

```
Struct “name”
{
        Position
        Normal
        Color
        TexCoords
        etc
}
```

Vertex Shader

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            ....
             VertexShaderOutput vert(VertexShaderInput v)
            {
                VertexShaderOutput o;

                o.pos = mul(UNITY_MATRIX_MVP, v.vertex);

                return o;
            }

            ENDCG
        }
```

Return type is VertexShaderOutput

‘v’ holds the input data coming from Unity

Here we do Model View Projection

Fragment Shader

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag
            ....
             float4 frag(VertexShaderOutput i):SV_TARGET
            {
                return float4(1, 0, 0, 1);
            }

            ENDCG
        }

}
```

Return type is float4

‘i’ holds the input data coming from the vertex shader

Return value is a color

Getting color from Unity

```
Shader "CM163/FirstShader"
{
    ....
        Pass
        {
                CGPROGRAM
                #pragma vertex vert
                #pragma fragment frag
                 float4 _Color;

                 float4 frag(VertexShaderOutput i):SV_TARGET
                {
                    return _Color;
                }

                ENDCG
        }

}
```

Define a uniform with same name as defined in properties