Introduction to Shader development

h_da WS2020/21

Paul Nasdalack

info@paul-nasdalack.com

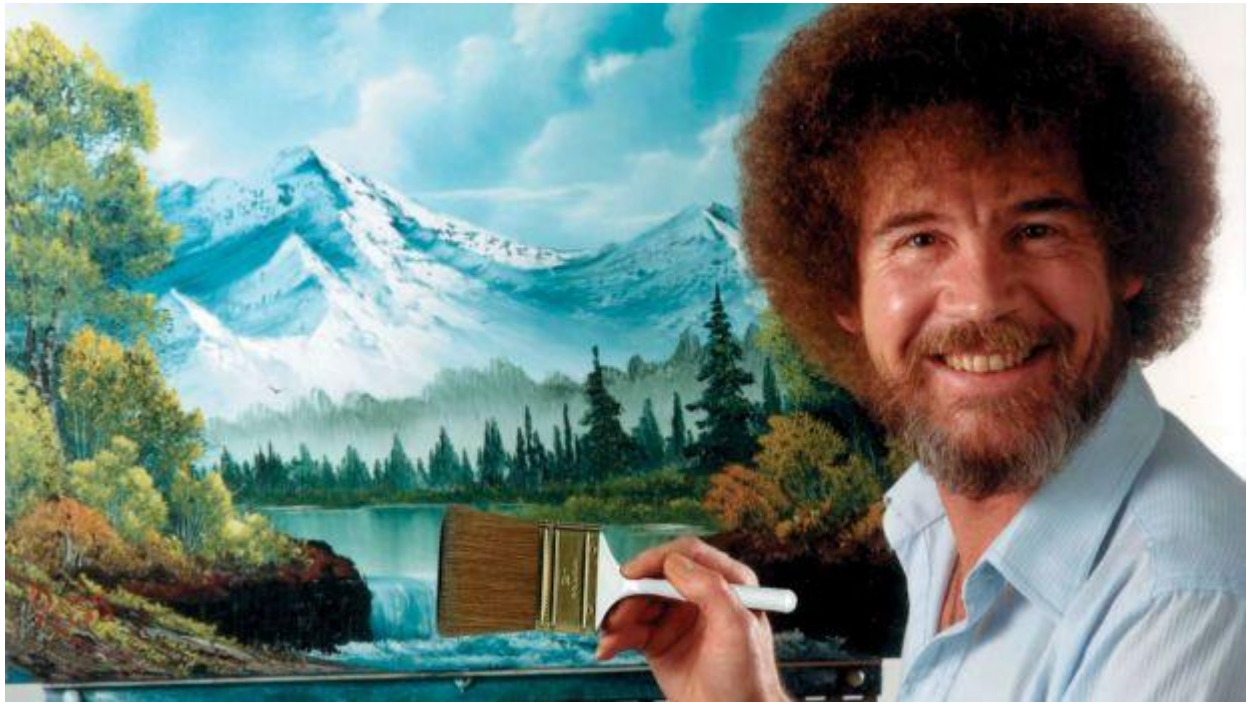
@littleBugHunter

Quick Recap

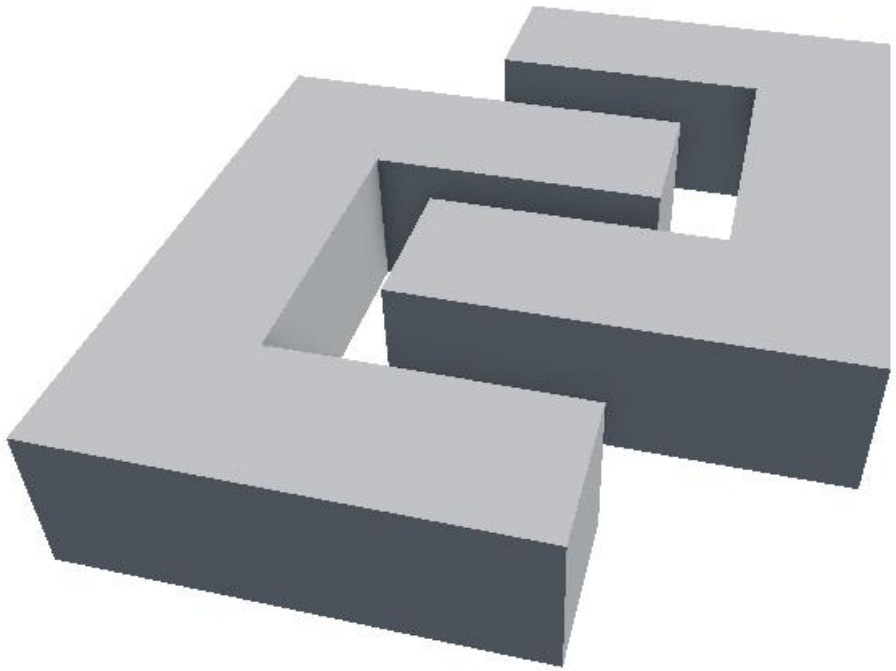
How does the GPU Render Objects

- What is in front and what is behind?
  - Painters Algorithm
  - Zbuffer Compare

Painters Algorithm

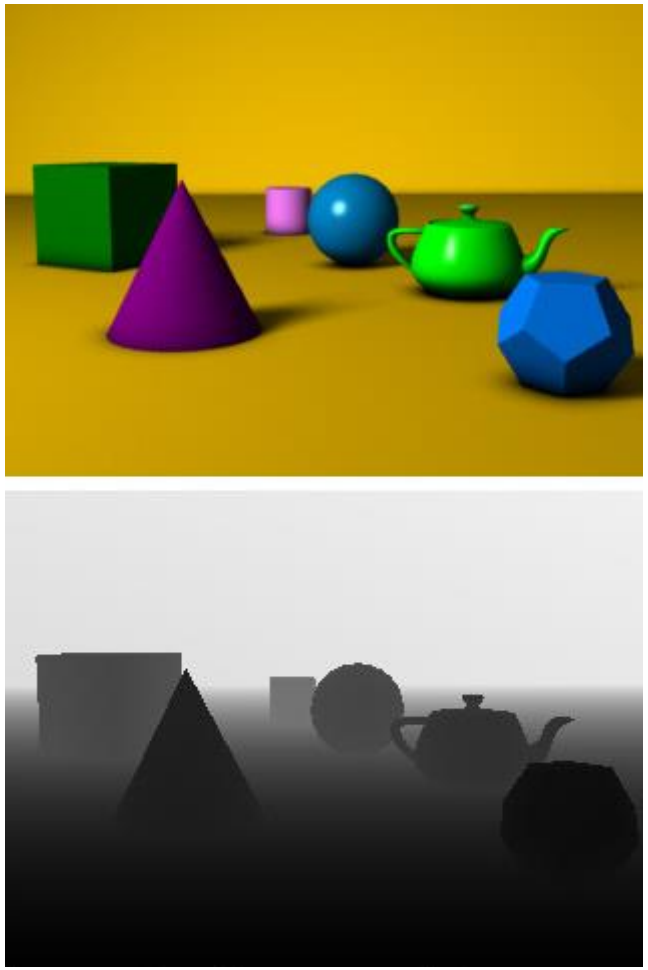


Painters Algorithm



ZBuffer

- Distance from the camera
- Stored in a second Texture
- Rasterizer checks each Pixels distance
- Resolved per pixel, not per object

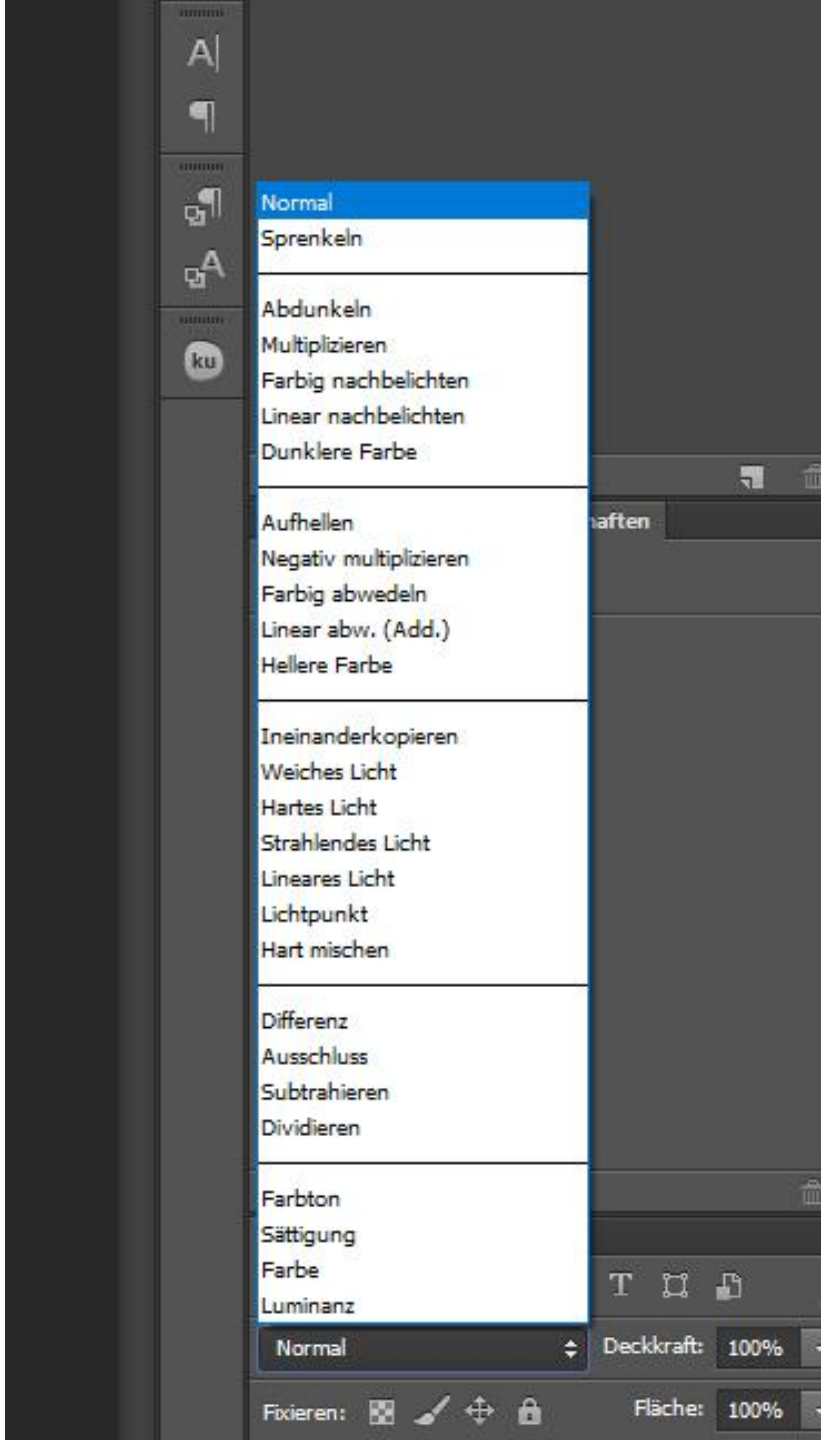


Blend Modes

- How to blend the object with the background
- Kind of like layer blend modes
- Always follow this formular:

  (You can look them up online)

```
NewColor*SrcBlend + OldColor*DstBlend
```

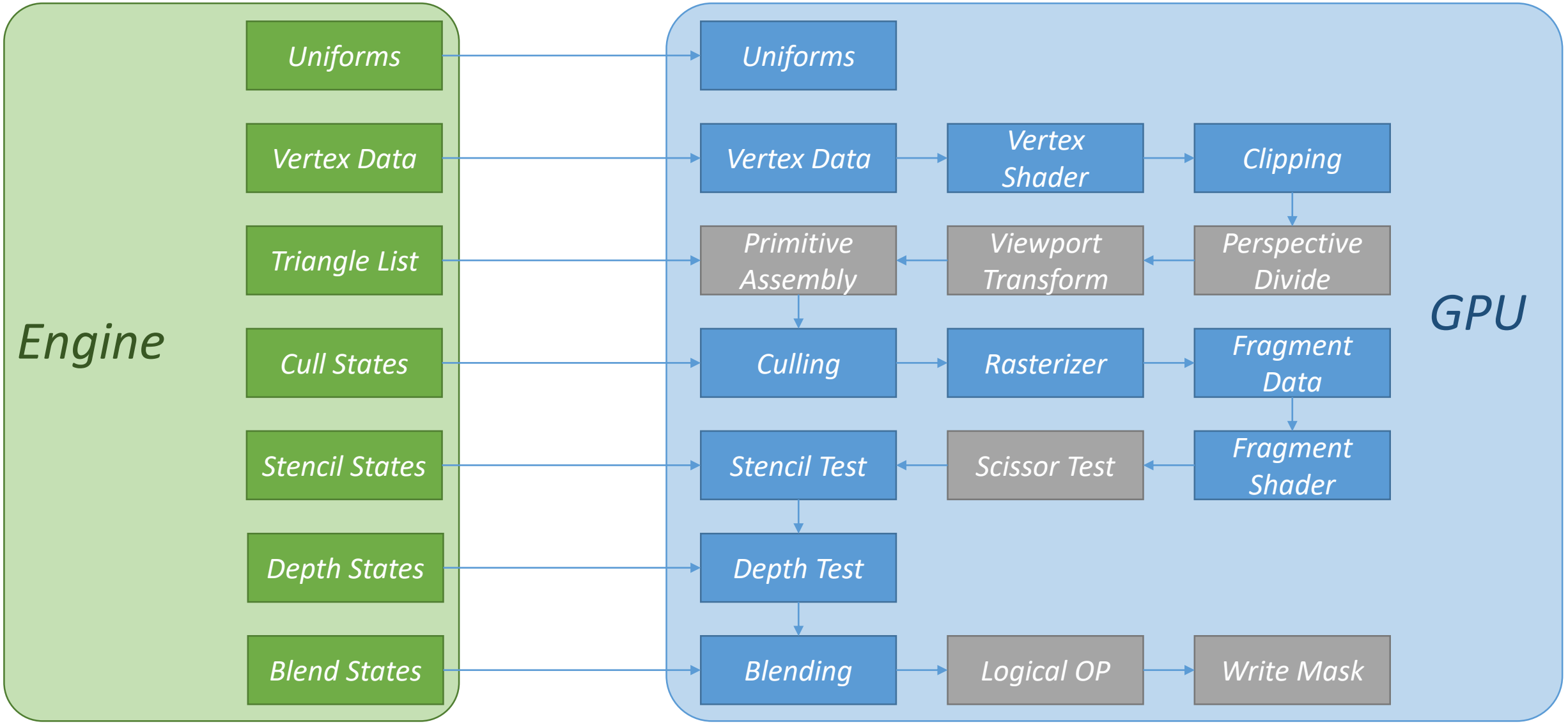


Congratulations!

- You basically know the entire Graphics Pipeline now
  - Direct X 9.0c that is
- Let’s have a rundown of the entire thing

https://www.opengl.org/wiki/Rendering_Pipeline_Overview

Graphics Pipeline



Current Data: Uniforms

Uniforms

- OpenGL concept (DX is a bit more complicated)
- Global Variables, set from the engine
- Unity Material Parameters for example
- Can not be changed by the shader

Current Data: Vertex Data

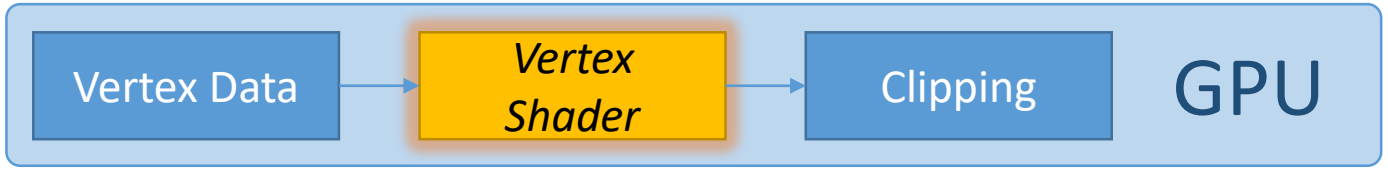
Vertex Data

- Our appdata Struct
- There is one struct for each Vertex in the Mesh
- Positions, Normals, UVs, Tangents, etc.
- Order is important!

Current Data: Vertex Data

Vertex Shader

Engine

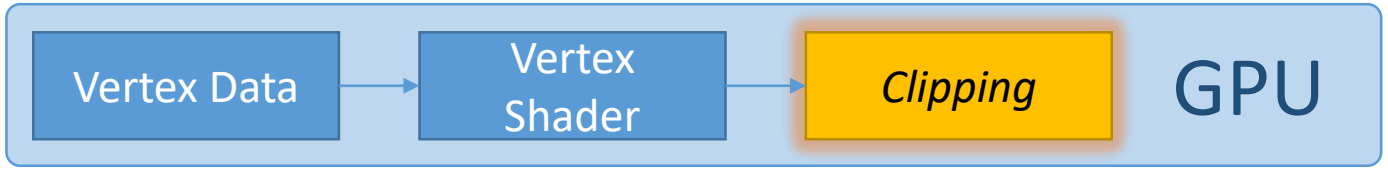


- First Shader
- Goes over each appdata struct and modifies it
- Typically moves vertices into screen space

Current Data: Vertex Data

Clipping

Engine

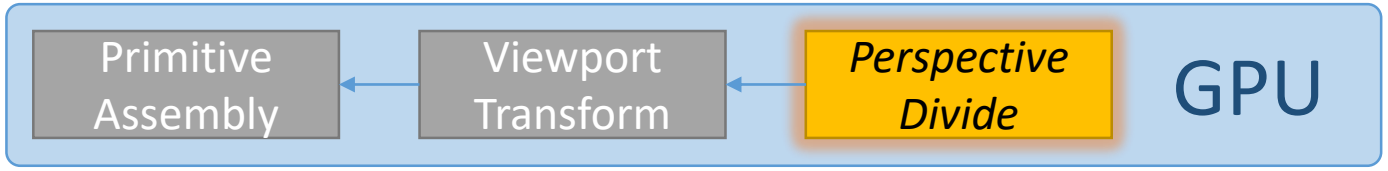


- All Vertices outside the view get thrown away, we don’t have to do any more calculations on them

Current Data: Vertex Data

Perspective Divide

Engine

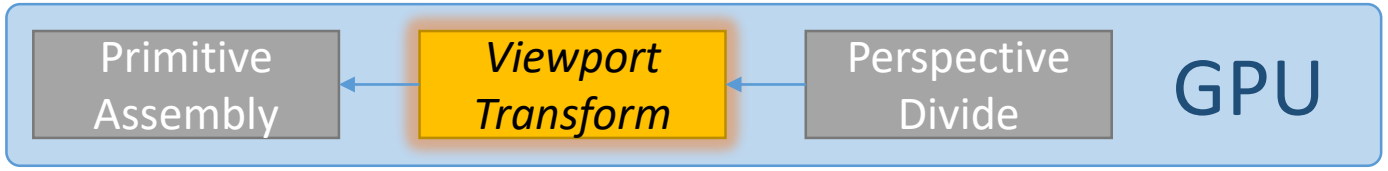


- Some Matrix Multiplications modify the 4^{th} component of a Vector
- we are in 3D space and need to renormalize the 4^{th} dimension
  - We actually divide every position by it's 4^{th} component

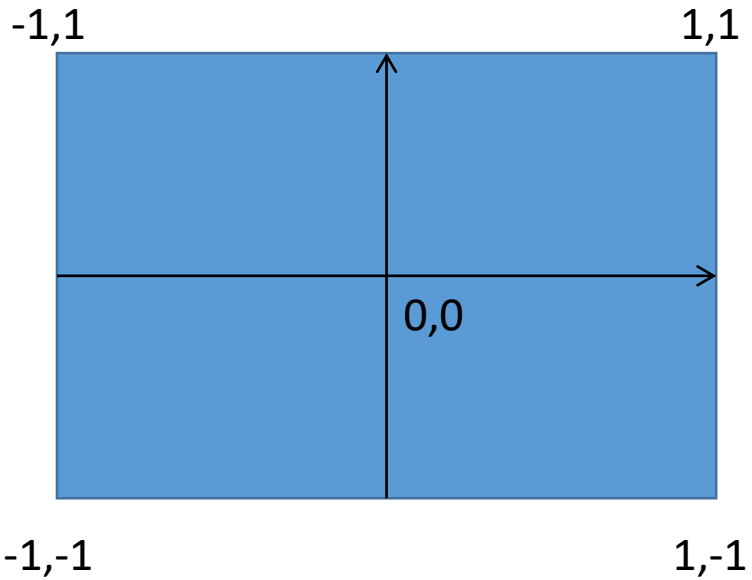
Current Data: Vertex Data

Viewport Transform

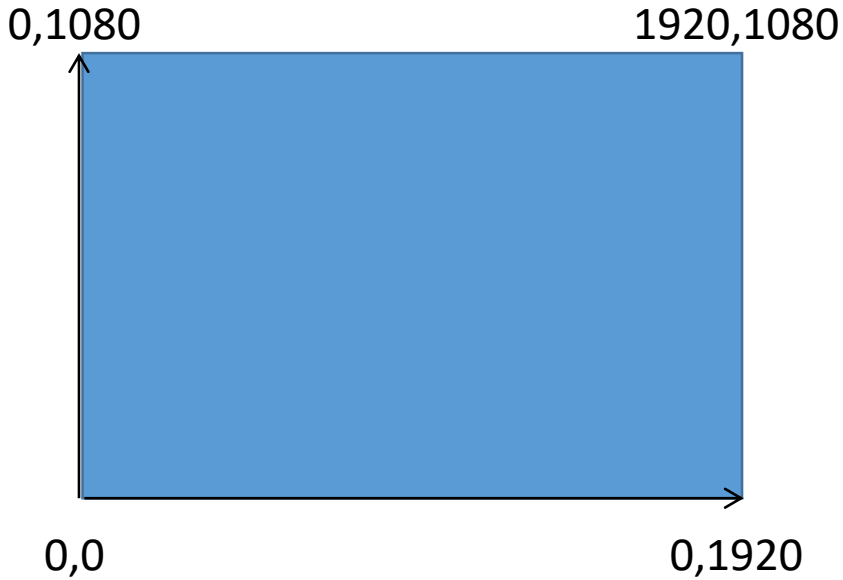
Engine



- The positions are moved from NDC space to actual window space
  - NDC = Normalized Device Coordinates



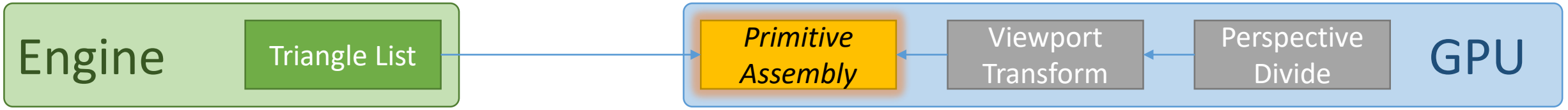
NDC



Window space

Primitive Assembly

Current Data: Vertex Data, Triangle List



- Assemble Triangles from the Triangle list
  - Vertices were pushed a specific order, so they can just be numbered
- Triangle list is just connecting numbered points :P

Culling

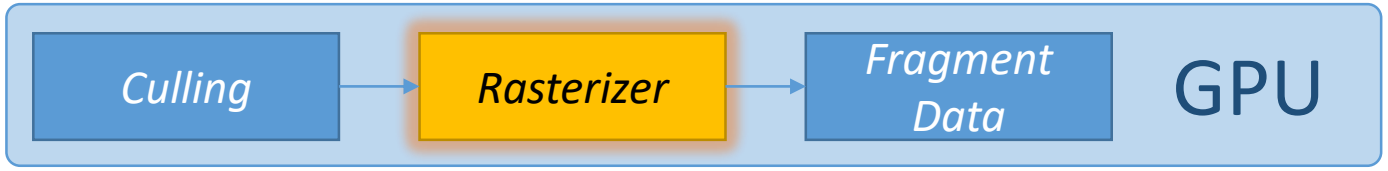
Current Data: Primitives (Triangles)

- Remove Triangles
  - Facing away from the camera
  - Facing to the camera
  - Or none at all
- Depending on Cull State

Current Data: Primitives (Triangles)

Rasterizer

Engine

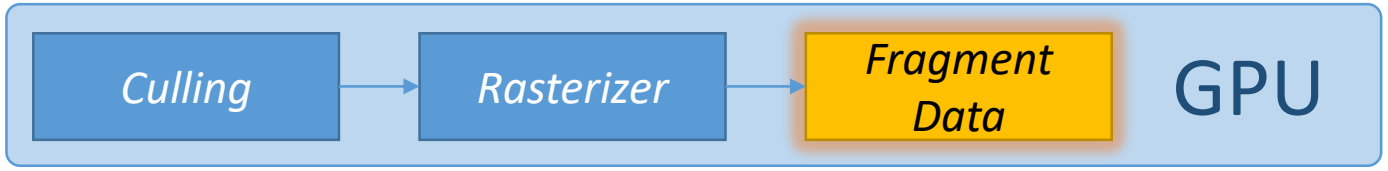


- Goes through each triangle
- Creates a Fragments for each pixel covered by it
- If multiple triangles are overlapping there will be multiple fragments per pixel

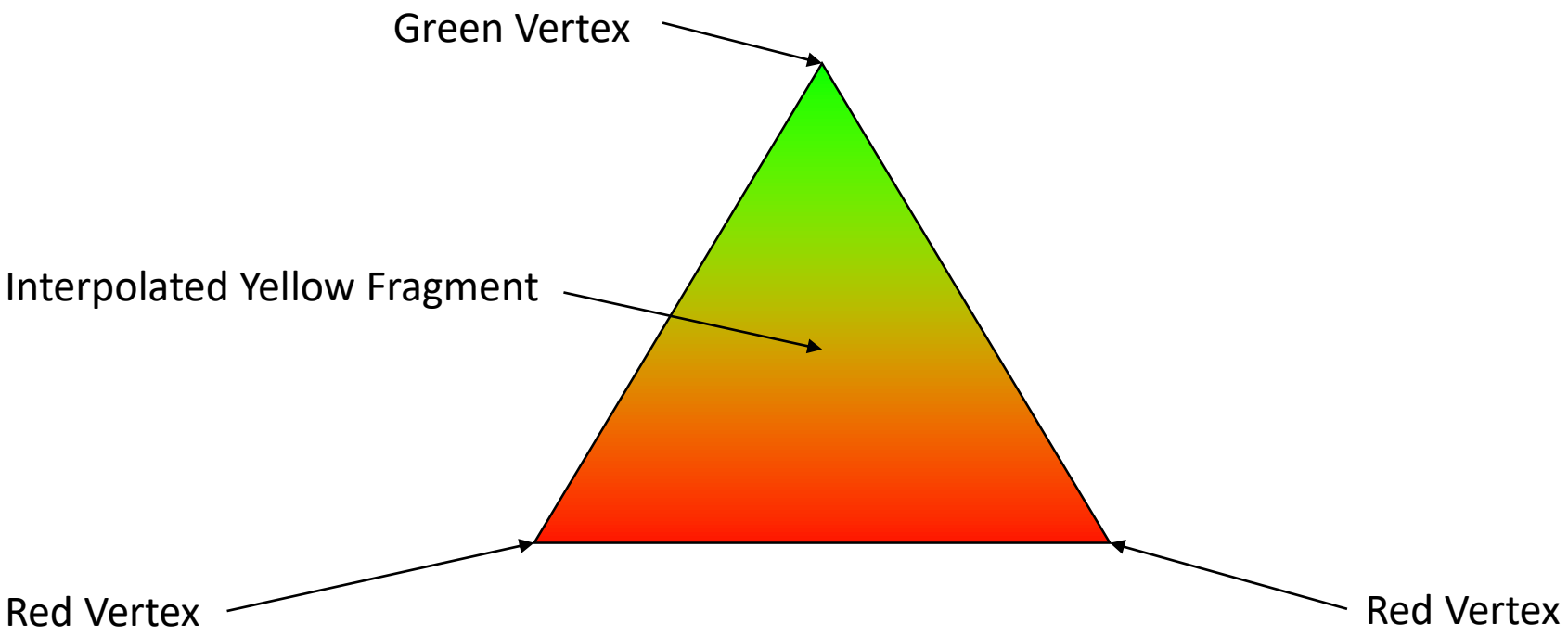
Current Data: Fragment Data

Fragment Data

Engine



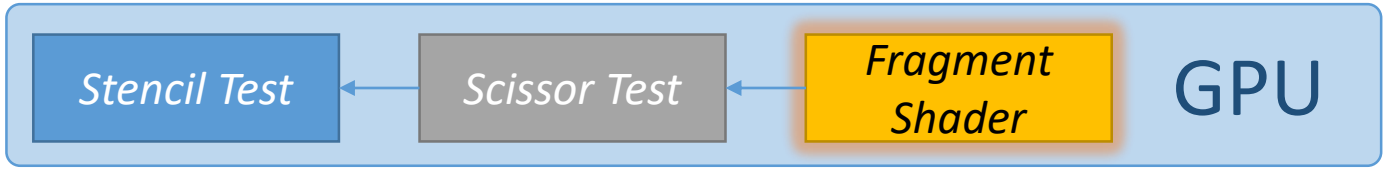
- Each Fragment contains interpolated vertex output data (v2f struct)



Current Data: Fragment Data

Fragment Shader

Engine

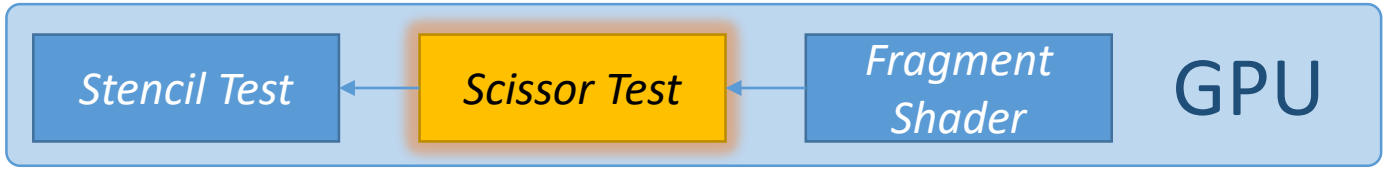


- Second Shader
- Runs on each Fragment
- Typically calculates Texturing, Lighting etc.
- Returns a Color

Scissor Test

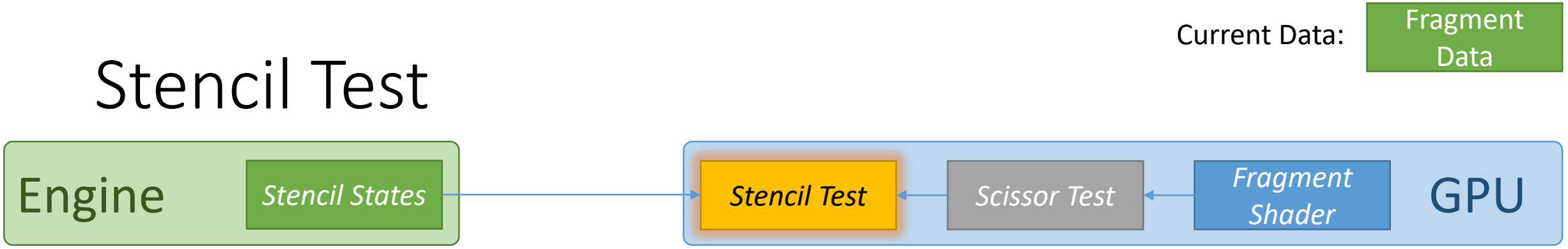
Current Data: Fragment Data

Engine



- Discards any Fragments outside the Scissor Rectangle
- Rectangle can be specified to cover a certain area of the screen
  - E.g. Split Screen
- Typically Scissor just covers the entire screen

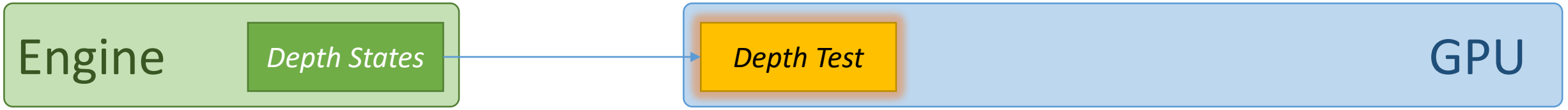
Stencil Test



- Special Buffer for custom user operation
- Stores integer values for each pixel ranging from 0-255
- Can be compared, incremented, decremented or set
- Used for various effects:
  - Portals
  - Dynamic holes
  - UI clipping
  - Etc.

Depth Test

Current Data: Fragment Data



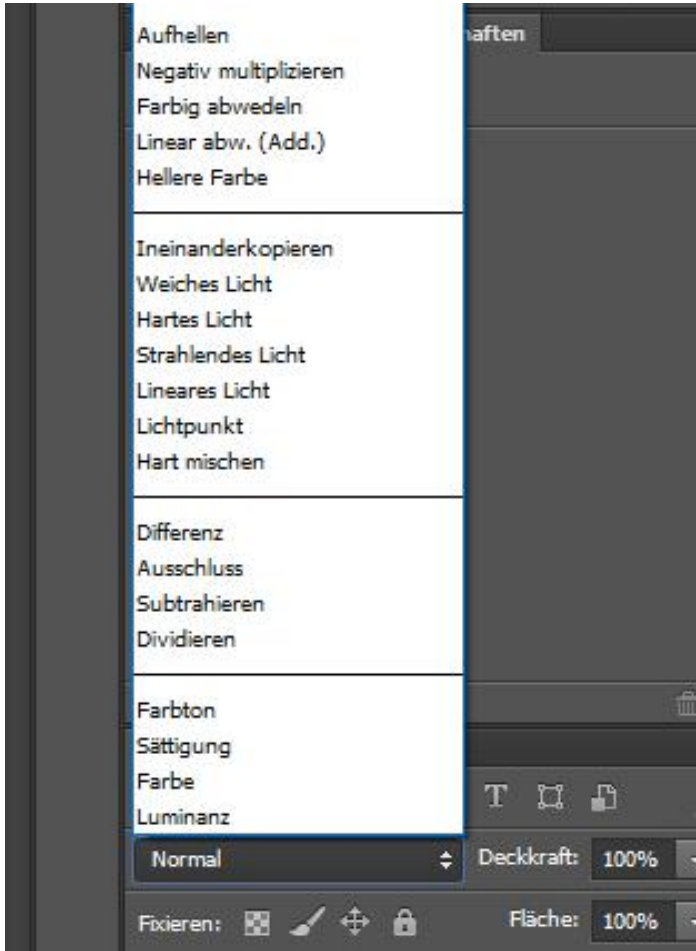
- Compares each fragment with the Depth buffer
  - Is it closer to the camera, than what we've already written?
- If comparison fails, fragments will be discarded
- If comparison succeeds, overrides the old Depth value

Blending

Current Data: Fragment Data

- Gets the old Color from the Screen (DstColor)
- Blends the new Value (SrcColor) with the old Value
- Uses this Formula:
  - (You can look up the values online)

```
SrcColor*SrcBlend + DstColor*DstBlend
```



Logical OP

Current Data: Fragment Data

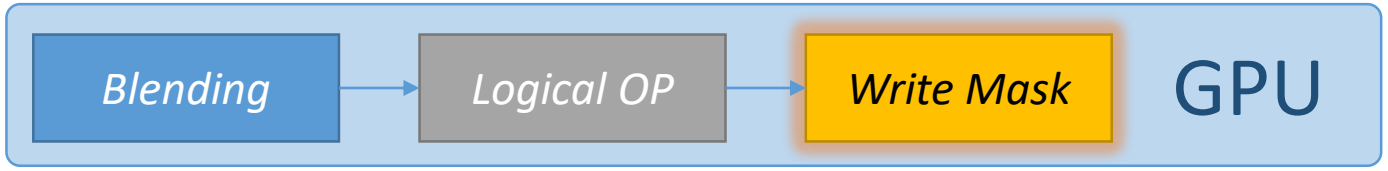
Engine

- Custom Bitwise Operations on Color
- Used rarely nowadays

Write Mask

Current Data: Fragment Data

Engine



- Can discard certain components of the Fragment
- Colors per channel
  - Eg. Only let red and blue pass and discard Green
- Depth
  - Disable Depth write for transparent objects
- Stencil Buffer with a bitmask for custom operations

Why?

Why?

- Every GPU out there follows this pipeline
- Every time you interact with the GPU to render something it will follow these steps
- Vendor agnostic (Nvidia, AMD, Intel all use this pipeline)
- Every PC, Mac, Phone, Console and Raspberry PI runs like this