Visualization and Rendering

Pipeline, Shaders, Materials

Tomáš Polášek ipolasek@fit.vutbr.cz

Brno University of Technology

Faculty of Information Technology
DCGM, CPhoto@FIT
Faculty of Fine Arts
Game Media Studio

7

Rendering Concepts

Rendering as Visualization

- Goal: Visualize the World Game World
- Image Synthesis
- Real-Time Graphics
- “Fake Everything”
- → Rendering Engine [1]

Source: Alen Keys, Corona / Dark Souls 3

The Scene

- Coordinates & Axes
- Observer
- Objects:
  - Representation
  - Visual Properties
- Light Sources
  - Types
  - Interactions
- Camera
  - Model
  - Projection
- Virtual Screen
- Rendered Image

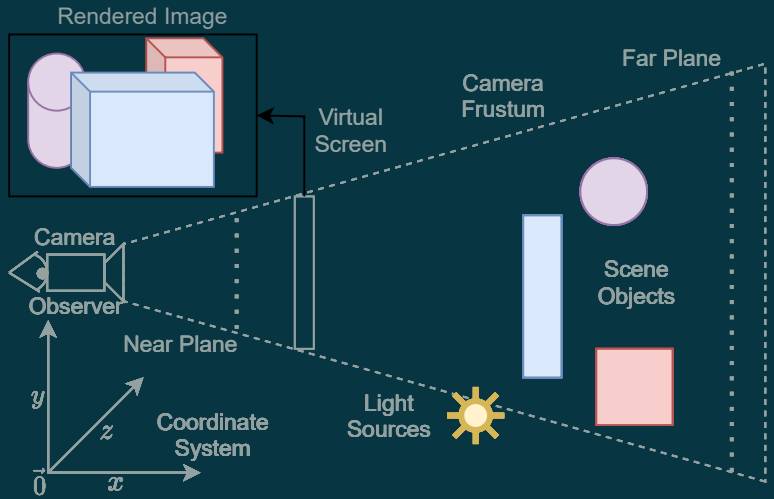
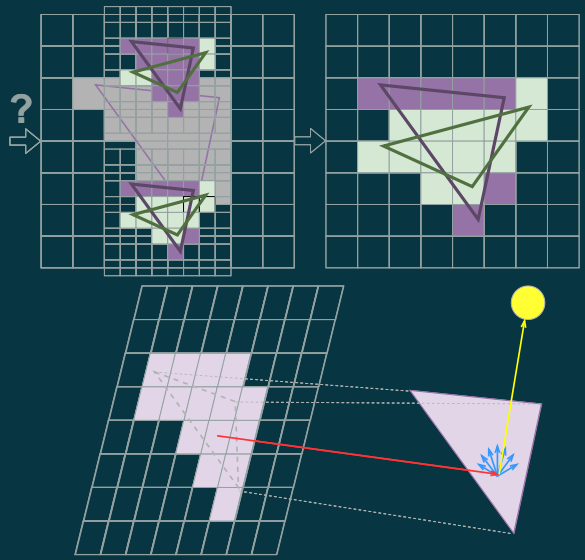


Image Synthesis

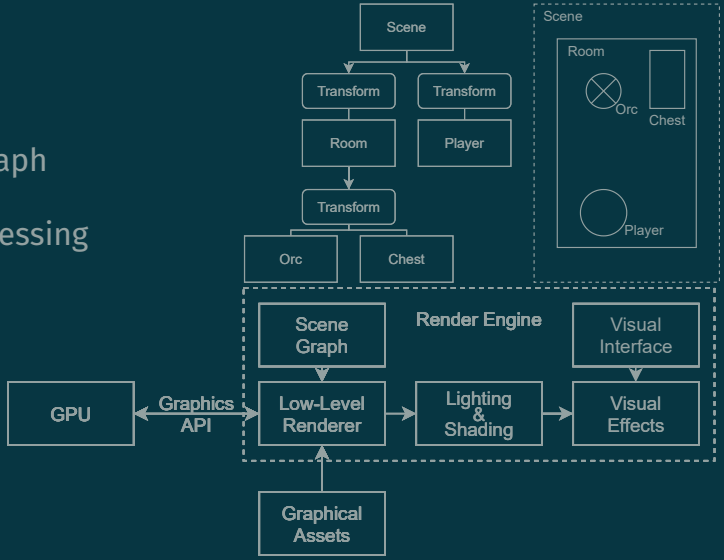
- **Scene → 2D Image**
- Image Raster
- Rendering Process
- Approaches
  - Rasterization + Depth
  - Ray Casting + Ray Tracing
- Shading & Lighting



Rendering Games

Engine Overview

- GPU → Graphics API
- Graphical Assets
- Low-Level Rendering
- Virtual World → Scene Graph
- Lighting & Shading
- Visual Effects & Post-Processing
- User Interface

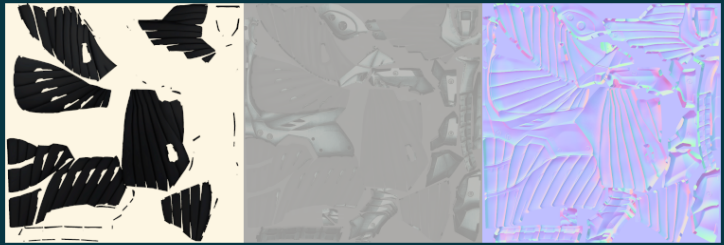


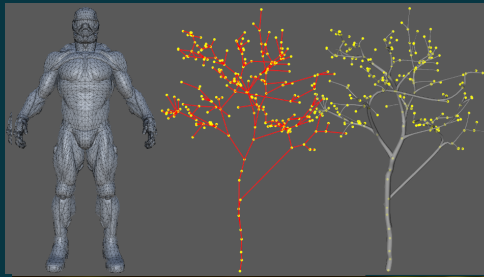
Graphics APIs

- GPU Communication
- Data & Memory
- Range of Options
- Varied Properties
  - Supported Platforms
  - Additional Functions
  - Abstraction Level

Graphical Assets

- Object Properties
- Models:
  - Curves
  - Polygonal Meshes
  - Graphs
  - Distance Functions
- Materials & Textures





Source: Inigo Quilez: Ladybug

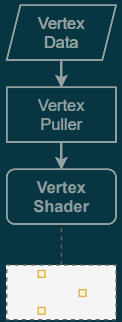
Low-Level Renderer

- Rendering Primitives
- GPU Architecture & Parallelism
- Fixed vs Programmable
  → Shaders
- General Purpose GPU
- Rendering Pipeline [2]

Source: NVIDIA

Vertex Processing

- Vertex Properties & Assembly
- Vertex Shader
- Model Space → World Space
- Per-Vertex Shading

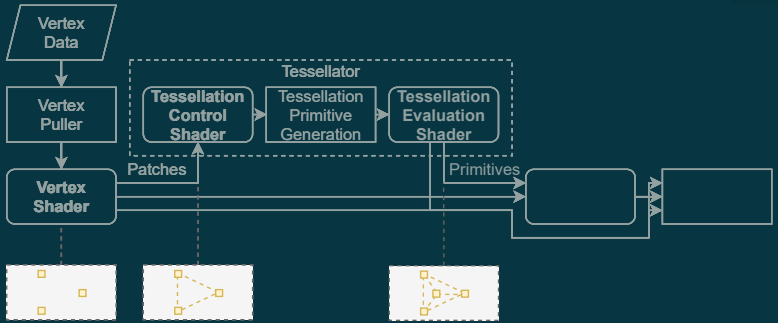



```
layout (location=0) in vec3 iPosition;
layout (location=1) uniform mat4 uMVP;
out vec4 vColor;

void main()
{ // Vertex Shader:
    gl_Position = uMVP * vec4(iPosition, 1.0);
    vColor = vec4(1.0, 0.0, 1.0, 1.0);
}
```

Patch Tessellation

- Subdivide Patches
- Control Magnitude
- Evaluate Positions
- Finer Meshes

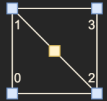
```
void main()
{ // Tessellation Control Shader:
    gl_TessLevelOuter = float[4](4, 1, 6, 1);
    gl_TessLevelInner = float[2](5, 1);
}

void main()
{ // Tessellation Evaluation Shader:
    gl_Position = uMVP * vec4(
        iPosition * gl_TessCoord, 1.0);
}
```

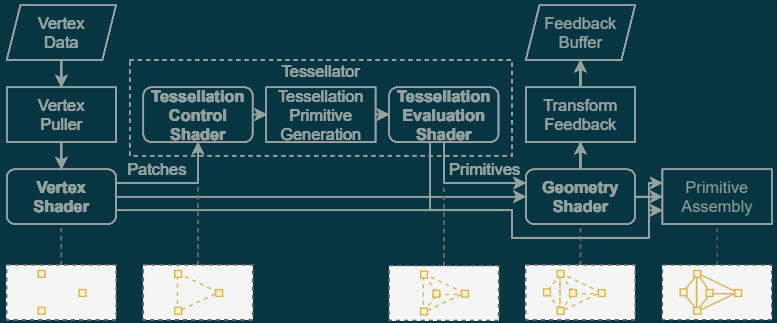


Primitive Geometry

- Geometry Modification
- Primitive Generation
- Feedback Buffer
- Assembly of Primitives

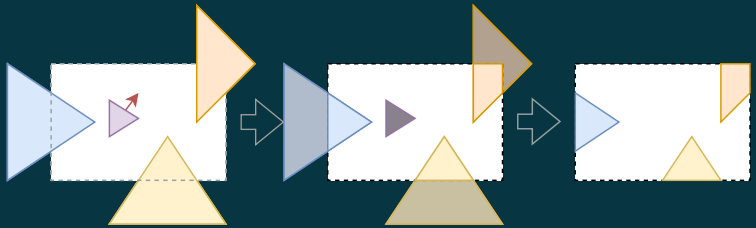
```
layout (points) in;
layout (triangle_strip, max_vertices=4) out;

void main()
{ // Geometry Shader:
    vec3 basePos = gl_in[0].gl_Position.xyz;
    gl_Position = basePos + vec3(-1.0f, -1.0f, 0.0f); EmitVertex();
    gl_Position = basePos + vec3(-1.0f,  1.0f, 0.0f); EmitVertex();
    gl_Position = basePos + vec3( 1.0f, -1.0f, 0.0f); EmitVertex();
    gl_Position = basePos + vec3( 1.0f,  1.0f, 0.0f); EmitVertex();
    EndPrimitive();
}
```

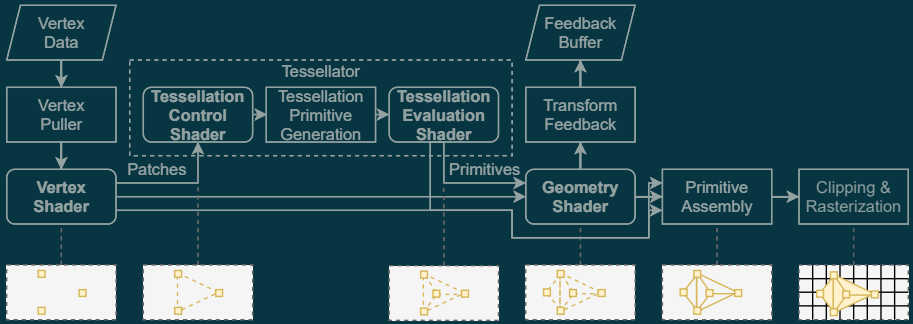




Clipping and Rasterization

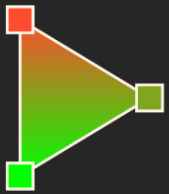
- Culling & Clipping
- Rasterization
- Early Z-Test
- Fragment Selection



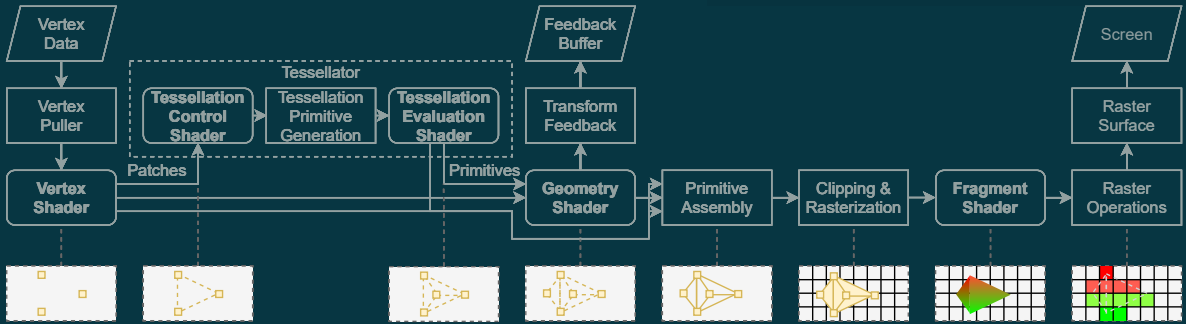


Fragment Operations

- Per-Fragment Processing
- Fragment Shader
- Vertex Interpolation
- Rasterized Pixels → Framebuffer



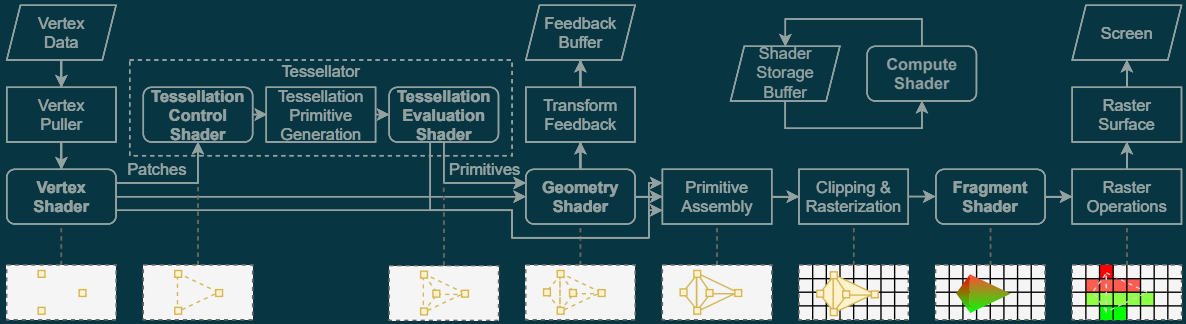
```
smooth in vec4 vColor;
out vec4 fColor;

void main()
{ // Fragment Shader:
    fColor = vertexColor;
}
```



General Purpose GPU

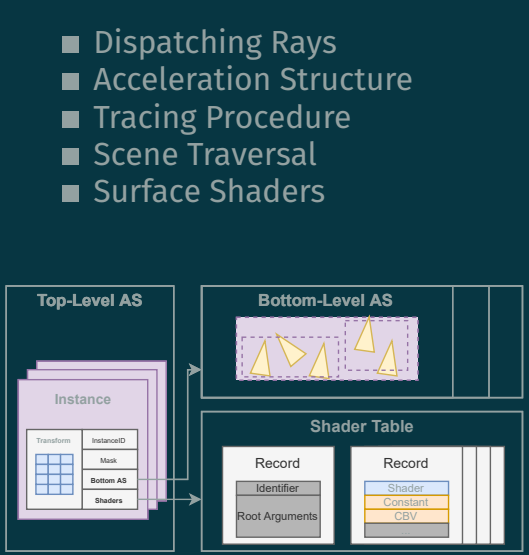
- Uses: Visual Effects
- Data on GPU
- Compute Shader
- Compute Dispatch

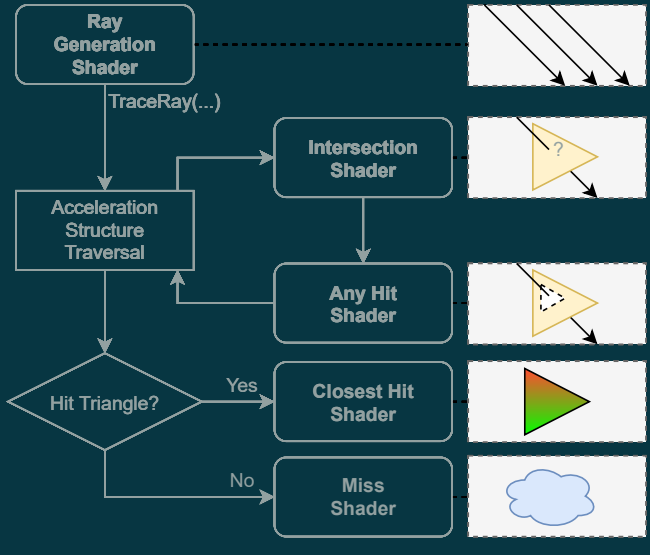
```
layout (local_size_x=32,local_size_y=32,local_size_z=8) in;
layout (binding = 1) readonly buffer InBuffer{ mat4 data[]; } inData;
layout (binding = 2) buffer OutBuffer{ mat4 data[]; } outData;

void main()
{ // Compute Shader:
    uvec3 lId = gl_LocalInvocationID; uvec3 gSize = gl_WorkGroupSize;
    uint flatId = lId.z * gSize.x * gSize.y + lId.y * gSize.x + lId.x;
    outData.data[flatId] = usefulComputation(inData.data[flatId]);
}
```



Ray Tracing Pipeline

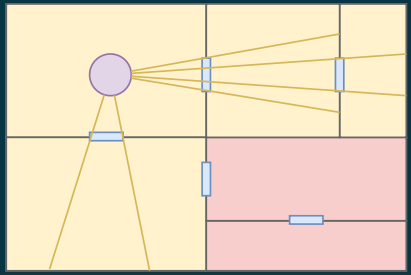
- Dispatching Rays
- Acceleration Structure
- Tracing Procedure
- Scene Traversal
- Surface Shaders

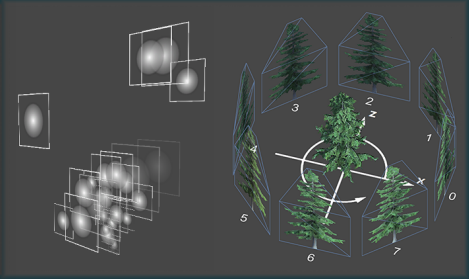




Optimizing Rendering

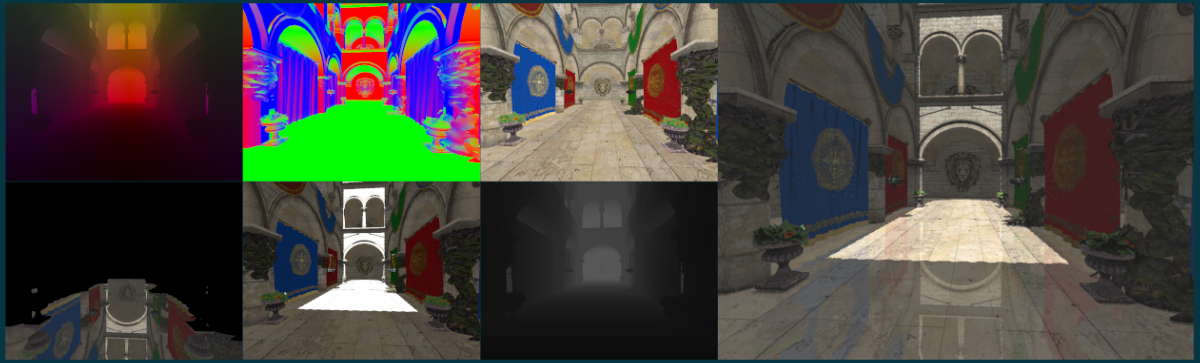
- Scene Graph
- Occlusion Culling
- Potentially Visible Set
- Billboards & Impostors
- Level of Detail & Proxies





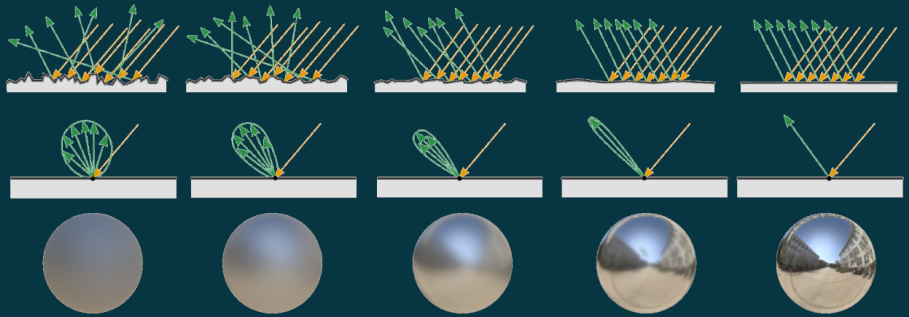
Deferred Rendering

- G-Buffer
- Render Pass → Composition
- Deferred Shading & Lighting
- Tiles & Clustered Rendering

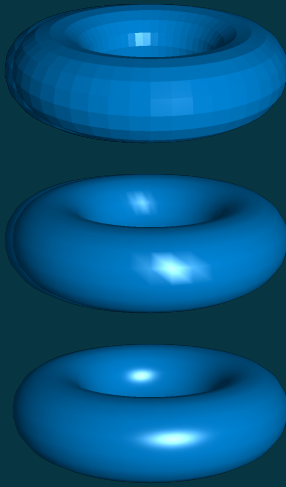


Lighting and Shading

- Shading Model: Flat, Gouraud, Phong
- Lighting & Shadowing
- Material Models → Blinn-Phong & GGX
- Surface Properties → BRDF



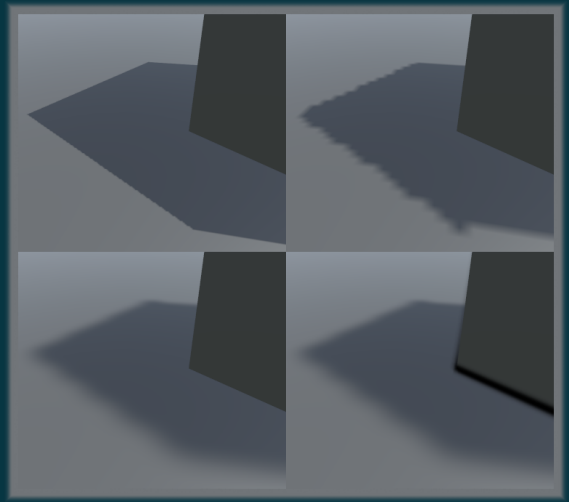
Source: Moving Frostbite to Physically Based Rendering



Source: Maarten Everts: Shading

Graphical Effects

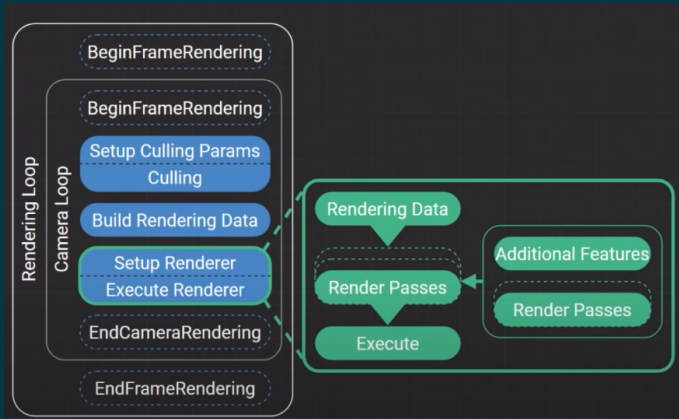
- Shadows & Soft Shadows
- Emulating Fidelity
  - Ambient Occlusion
  - Fog & Weather Effects
  - Normal, Bump, Parallax
- Visual Effects
  - Animation
  - Particles
  - Procedural
- Post Processing
  - Depth of Field
  - Motion Blur
  - Screen-Space Effects



Rendering in Unity

Unity Render Pipeline

- Standard → Black Box
- Scriptable Render Pipeline
  - C# Rendering API layer
  - Extensible with Sources
  - Shader Graph
  - VFX Graph
  - Post Processing Volumes
- Pre-Built: URP & HDRP
- Choosing the Solution



URP and HDRP

- UniversalRender Pipeline
  - “Wide” Support
  - Performance & Customizability
  - 2D Renderer Included
- High Definition Render Pipeline
  - Focused on High-End
  - Realistic Graphics

Source: Unity Pipeline Comparison

Additional Resources

- [Blog] Graphics Studies Compilation : DOOM 2016, DOOM Eternal
- [YouTube] Unity HDRP Demos The Heretic, Book of the Dead, Fontainebleau
- [UnityDoc] Choosing Render Pipeline
- [Blog] Inigo Quilez: Fractals, Computer Graphics, Shaders
- [Tutorial] Patricio G. Vivo: The Book of Shaders

Source: Inigo Quilez: Procedural Rainforest


Thanks For
Your Attention!
Far Cry

References I

[1] Jason Gregory. ***Game Engine Architecture, Second Edition.*** 3rd. USA: A. K. Peters, Ltd., CRC Press, 2018. ISBN: 1351974288.

[2] Khronos. ***Rendering Pipeline Overview.*** https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview. 2021.