

VISUALIZATION AND RENDERING

PIPELINE, SHADERS, MATERIALS

TOMÁŠ POLÁŠEK IPOLASEK@FIT.VUT.CZ

BRNO UNIVERSITY OF TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY

DCGM, CPhoto@FIT

FACULTY OF FINE ARTS

GAME MEDIA STUDIO



RENDERING CONCEPTS

RENDERING AS VISUALIZATION

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



Source: Aleš 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

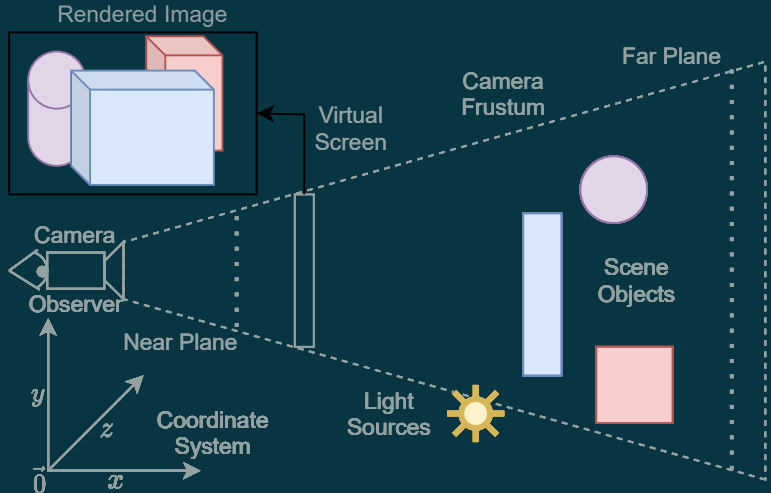


IMAGE SYNTHESIS

- **Scene → 2D Image**

- Image Raster

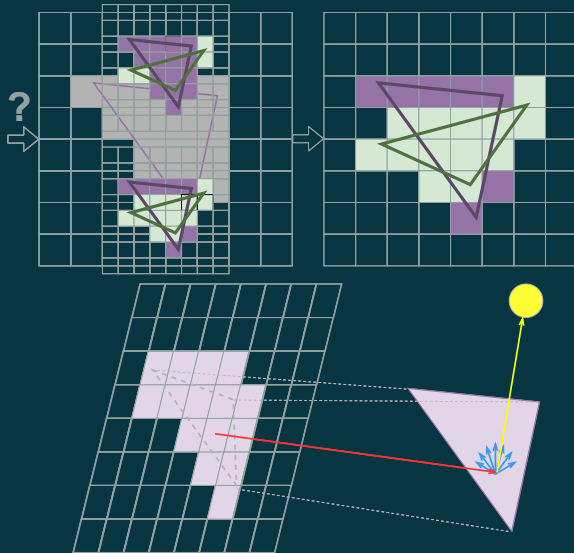
- Rendering Process

- Approaches

- Rasterization + Depth

- Ray Casting + Ray Tracing

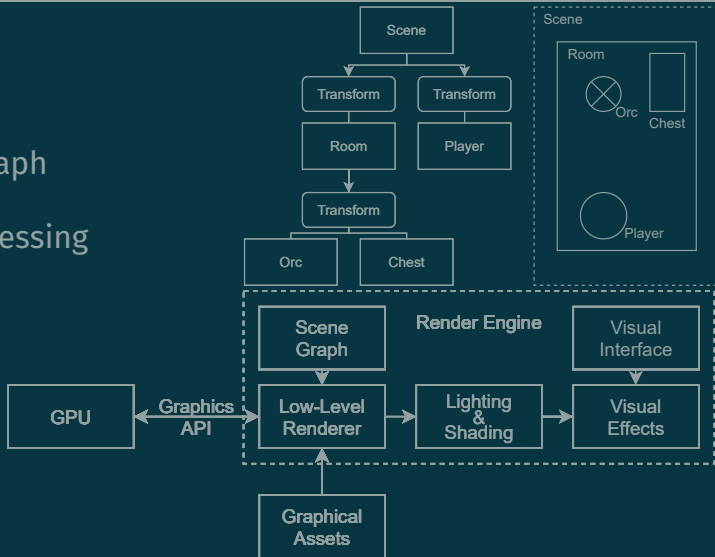
- Shading & Lighting



RENDERING GAMES

ENGINE OVERVIEW

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



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



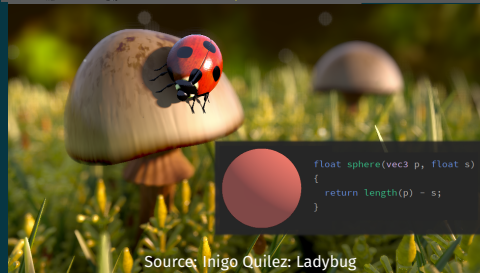
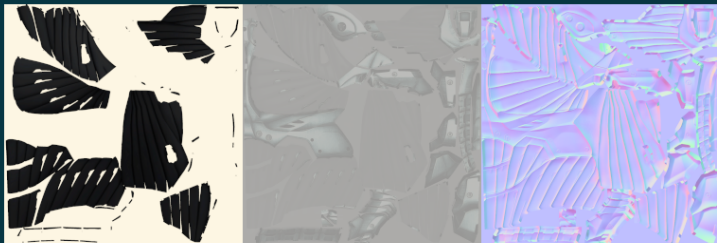
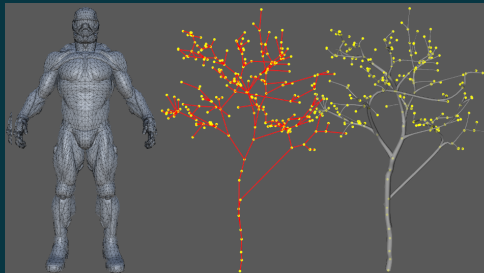
GRAPHICAL ASSETS

■ Object Properties

■ Models:

- ▶ Curves
- ▶ Polygonal Meshes
- ▶ Graphs
- ▶ Distance Functions

■ Materials & Textures



Source: Inigo Quílez: Ladybug

LOW-LEVEL RENDERER

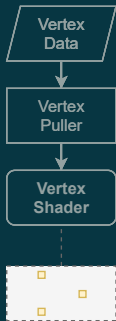
- Rendering Primitives
- GPU Architecture & Parallelism
- Fixed vs Programmable
 - Shaders
- General Purpose GPU
- Rendering Pipeline [2]



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);  
}
```



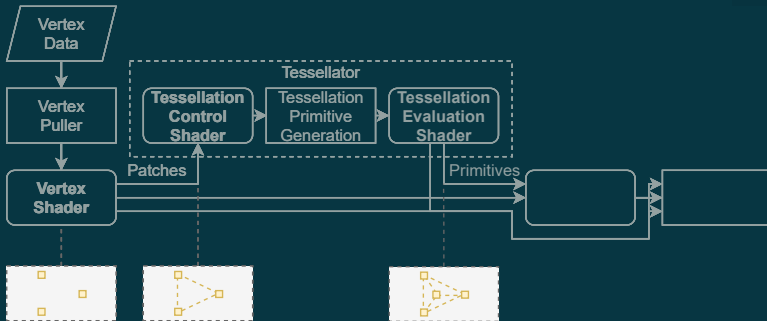
PATCH TESSELLATION

- 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);
}
```

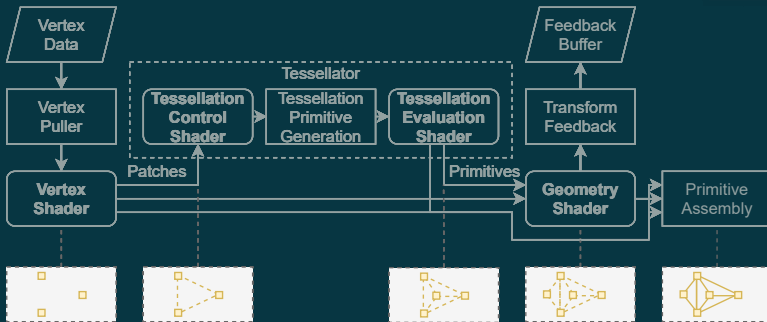
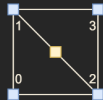
```
void main()
{ // Tessellation Evaluation Shader:
  gl_Position = uMVP * vec4(
    iPosition * gl_TessCoord, 1.0);
}
```



PRIMITIVE GEOMETRY

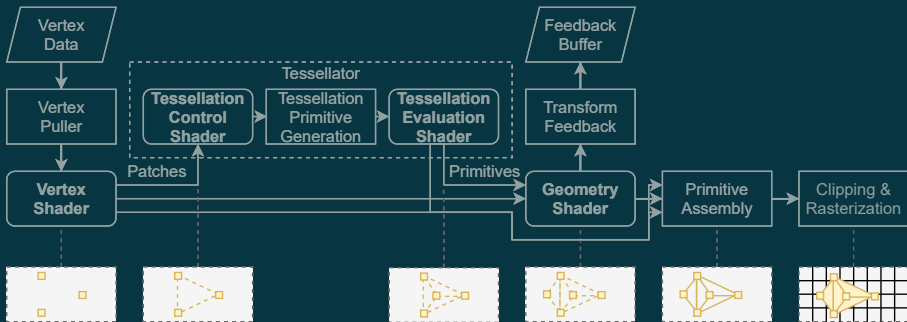
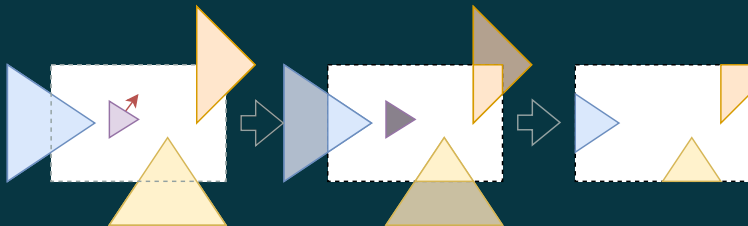
- Geometry Modification
- Primitive Generation
- Feedback Buffer
- Assembly of Primitives

```
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();  
}
```



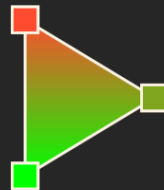
CLIPPING AND RASTERIZATION

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

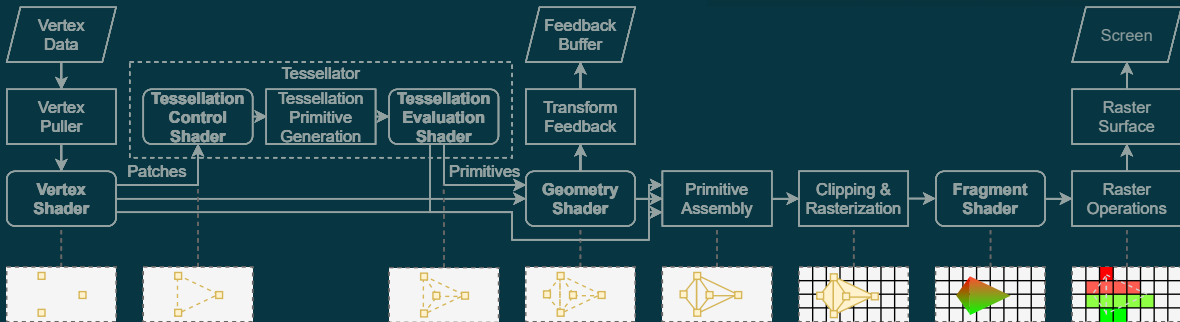


FRAGMENT OPERATIONS

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



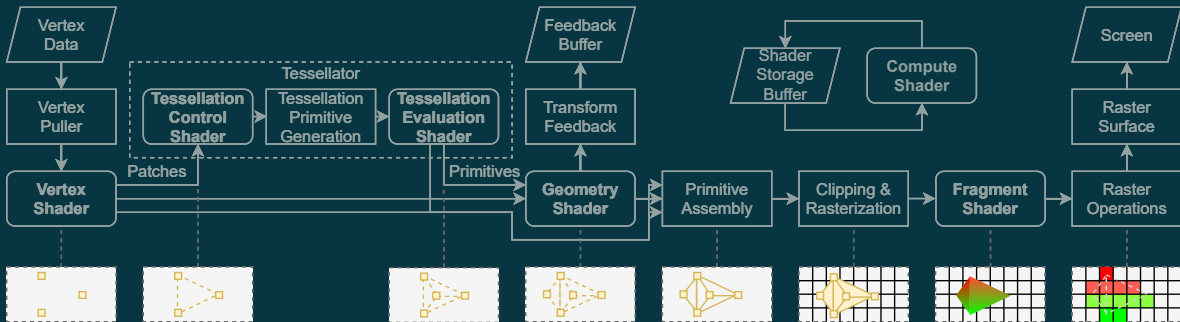
```
smooth in vec4 vColor;  
out vec4 fColor;  
  
void main()  
{ // Fragment Shader:  
    fColor = vColor;  
}
```



GENERAL PURPOSE GPU

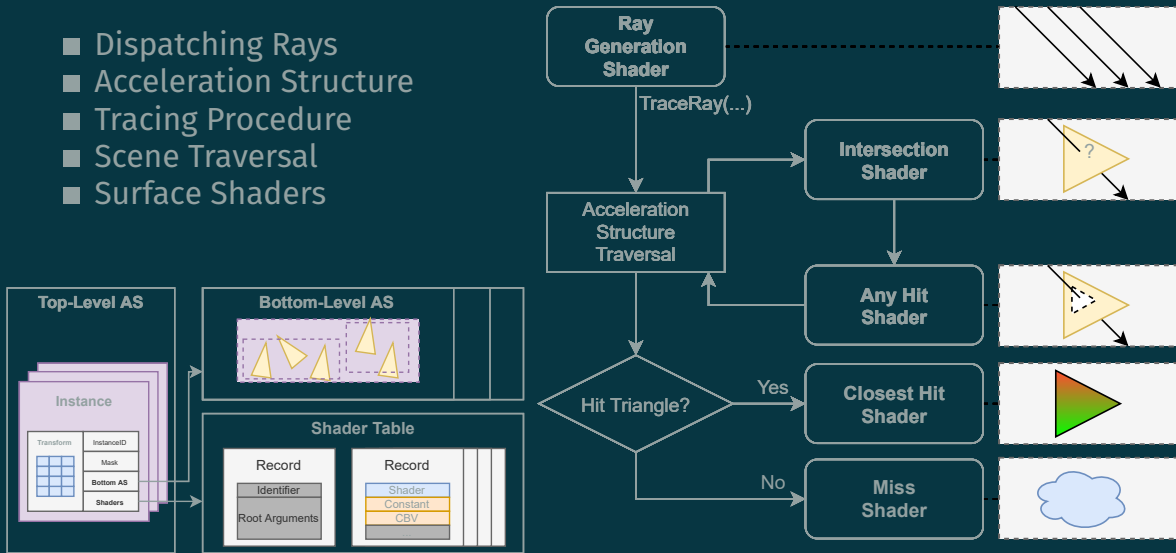
- Uses: Visual Effects
- Data on GPU
- Compute Shader
- Compute Dispatch

```
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]);  
}
```



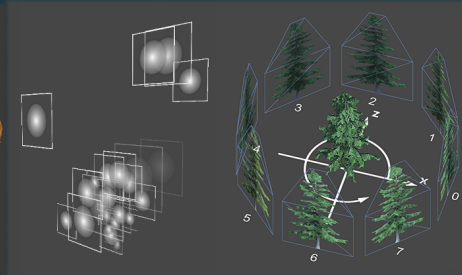
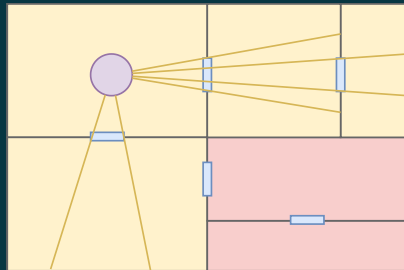
RAY TRACING PIPELINE

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



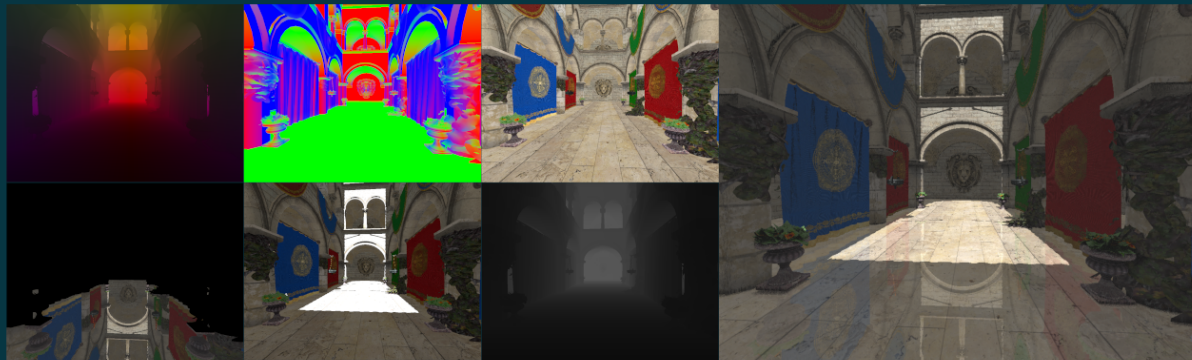
OPTIMIZING RENDERING

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



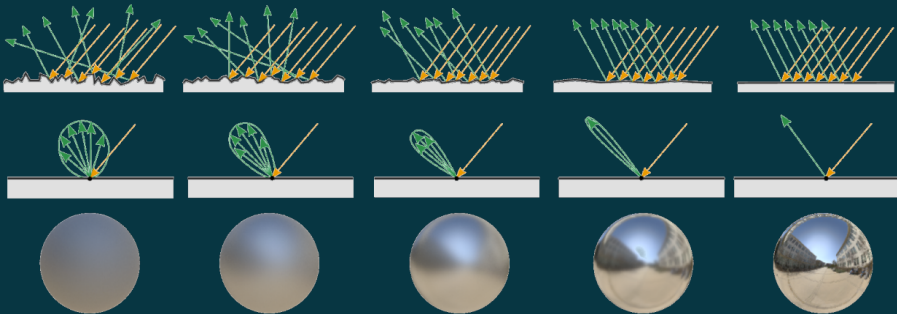
DEFERRED RENDERING

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

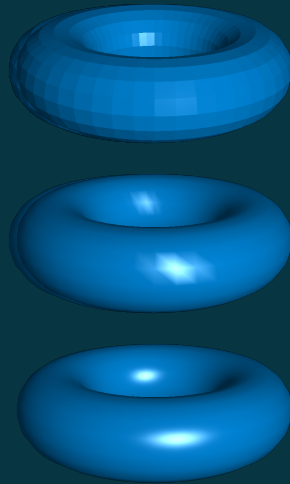


LIGHTING AND SHADING

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



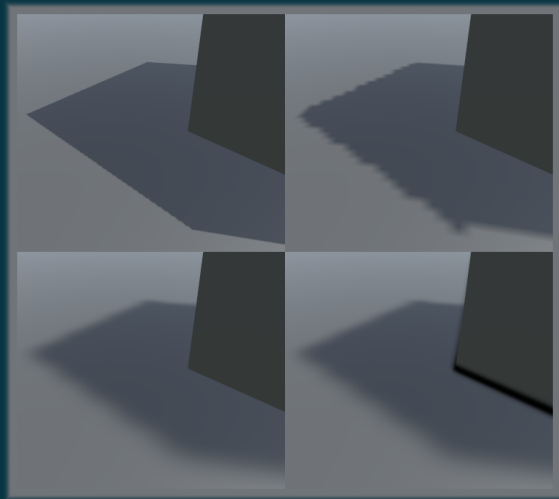
Source: Moving Frostbite to Physically Based Rendering



Source: Maarten Everts: Shading

GRAPHICAL EFFECTS

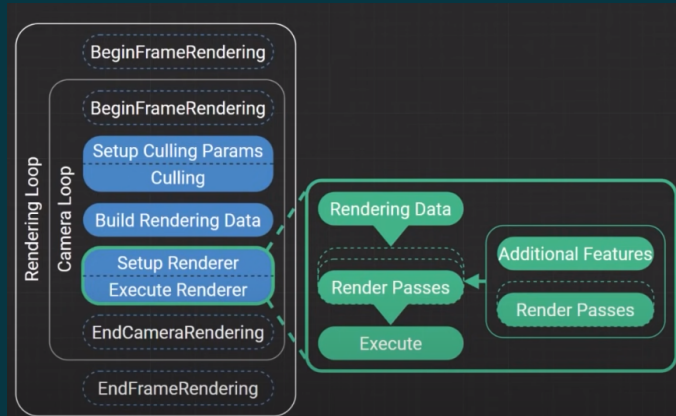
- Shadows & Soft Shadows
- Emulating Fidelity
 - Ambient Occlusion
 - Fog & Weather Effects
 - Normal, Bump, Parallax
- Visual Effects
 - Animation
 - Particles
 - Procedural
- Post Processing
 - Depth of Field
 - Motion Blur
 - Screen-Space Effects



RENDERING IN UNITY

UNITY RENDER PIPELINE

- Standard → Black Box
- Built-in, URP & HDRP
- Scriptable Render Pipelines
 - ▶ Examples in URP and HDRP
 - ▶ C# Rendering API layer
 - ▶ Extensible with Sources
 - ▶ Shader Graph
 - ▶ VFX Graph
 - ▶ Post Processing Volumes
- Choosing the Solution



URP AND HDRP

■ Universal Render Pipeline

- ▶ “Wide” Support
- ▶ Performance & Customizability
- ▶ 2D Renderer Included

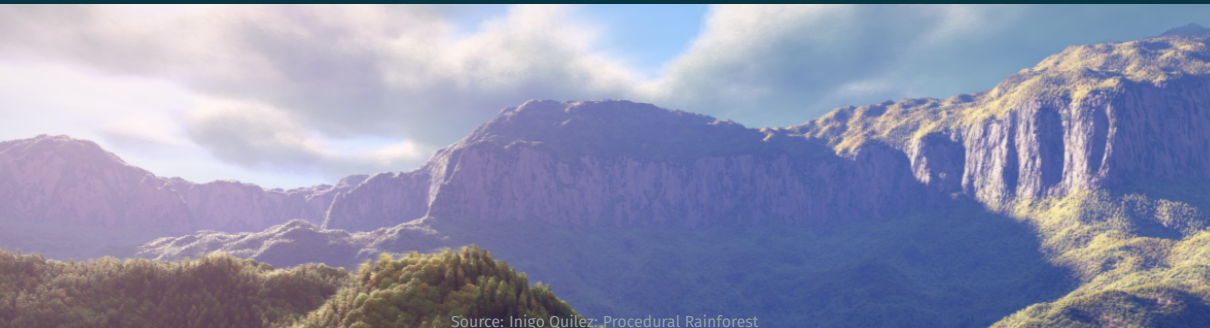
■ High Definition Render Pipeline

- ▶ Focused on High-End
- ▶ Realistic Graphics



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.