GAME ENGINE

Source

- The Cherno
- GAMES 104: Modern Game Engine – Theory and Practice

Framework

- Basic Elements
  - Structure and layer…
- Rendering
- Animation
- Physics
- Gameplay
  - Event system, scripts system, graph driven
- Misc. Systems
  - Effects, navigation, camera…
- Tool set
  - C++ reflection, data scheme (reflection: complex
- Online gaming
  - Synchronization, consistency

Advanced tech:
Motion matching
Procedural content generation (PCG

Data-oriented programming (DOP
Job system

(UE5 amazing systems)
Lumen
Nanite

Layered Architecture of Game Engine

- Tool Layer (chain of editors
- Function Layer (make it visible, movable and playable
- Resource Layer (data and file
- Core Layer (swiss knife of game engine
- Platform Layer (launch on different platforms

+Middleware and 3rd party libraries

Why:

- Decoupling and Reducing Complexity
- Response for Evolving Demands

Resource

how to access my data

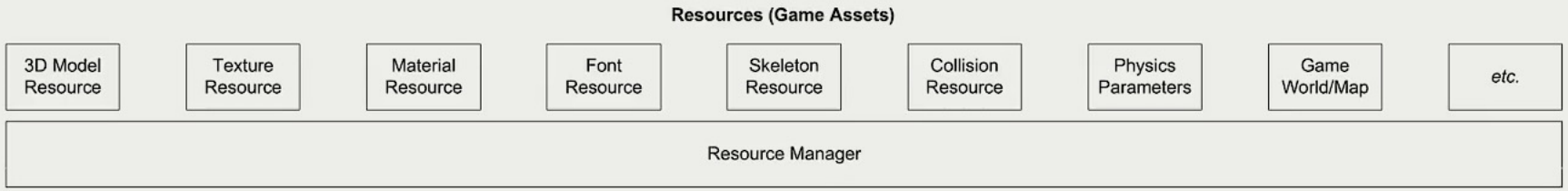
Offline Resource Importing

- Unify file access by defining a meta asset file format (ie.ast)
- Assets are faster to access by importing preprocess
- Build a composite asset file to refer to all resources
- GUID is an extra protection of reference

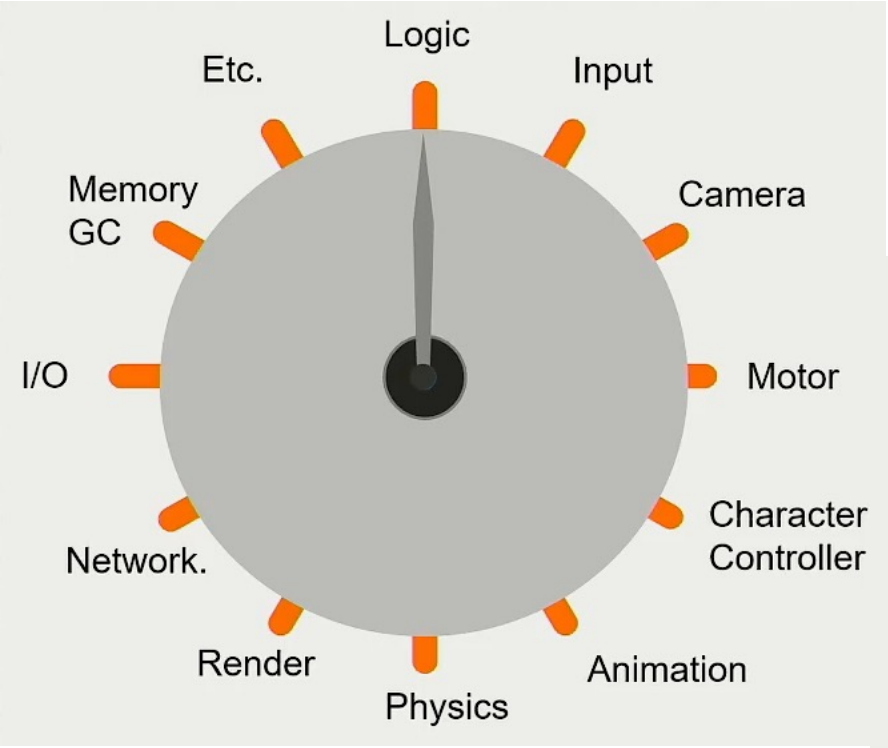
Manage asset life cycle

Memory management for Resources - life cycle

- Different resources have different life cycles
- Limited memory requires release of loaded resources when possible
- Garbage collection and deferred loading is critical features



Function



Tick
--logic
--render

Heavy-duty hotchpotch
Multi-threading

```
void tickMain(float delta_time)
{
    while (!exit_flag)
    {
        tickLogic(delta_time);
        tickRender(delta_time);
    }
}
```

```
void tickLogic(float delta_time)
{
    tickCamera(delta_time);
    tickMotor(delta_time);
    tickController(delta_time);
    tickAnimation(delta_time);
    tickPhysics(delta_time);
    /*...*/
}
```

```
void tickRender(float delta_time)
{
    tickRenderCamera();
    culling();
    rendering();
    postprocess();
    present();
}
```

Core

- Math Library
- Math Efficiency(quick and dirty hacks, SIMD
- Data structure and containers (customized STL
- Memory management (memory pool.

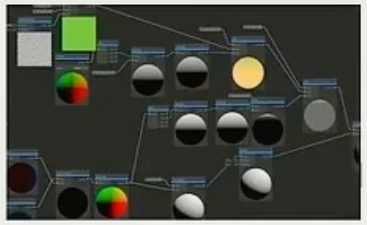
Platform

- File system
- Graphics API (DirectX, Vulkan…
- Hardware architecture

Tool

Level Editor

Logical Editor



Shader Editor

Animation Editor

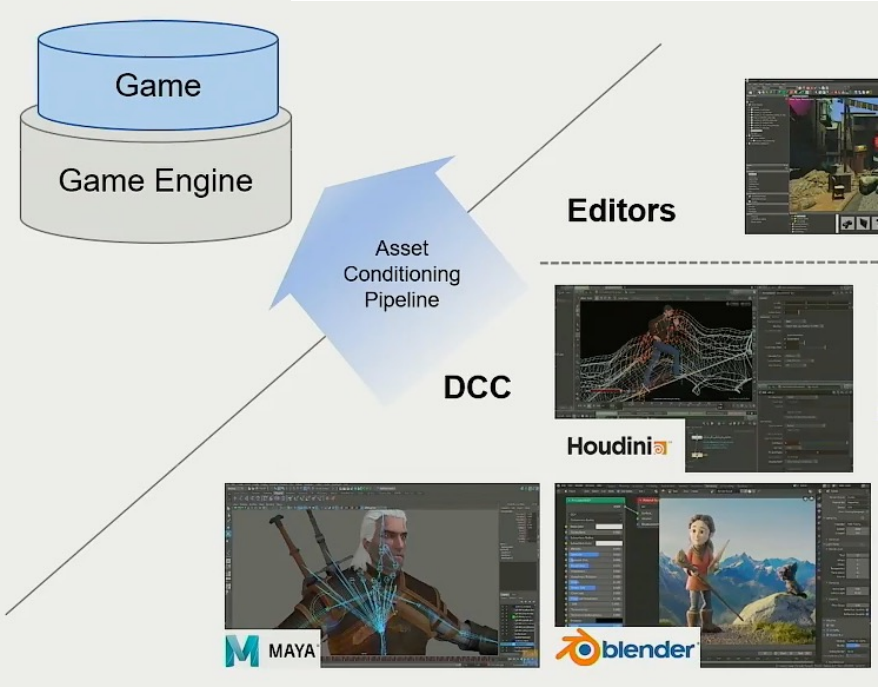
UI Editor

Allow everyone to create game

Unleash the Creativity

- Build upon game engine
- Create, edit and exchange game play assets

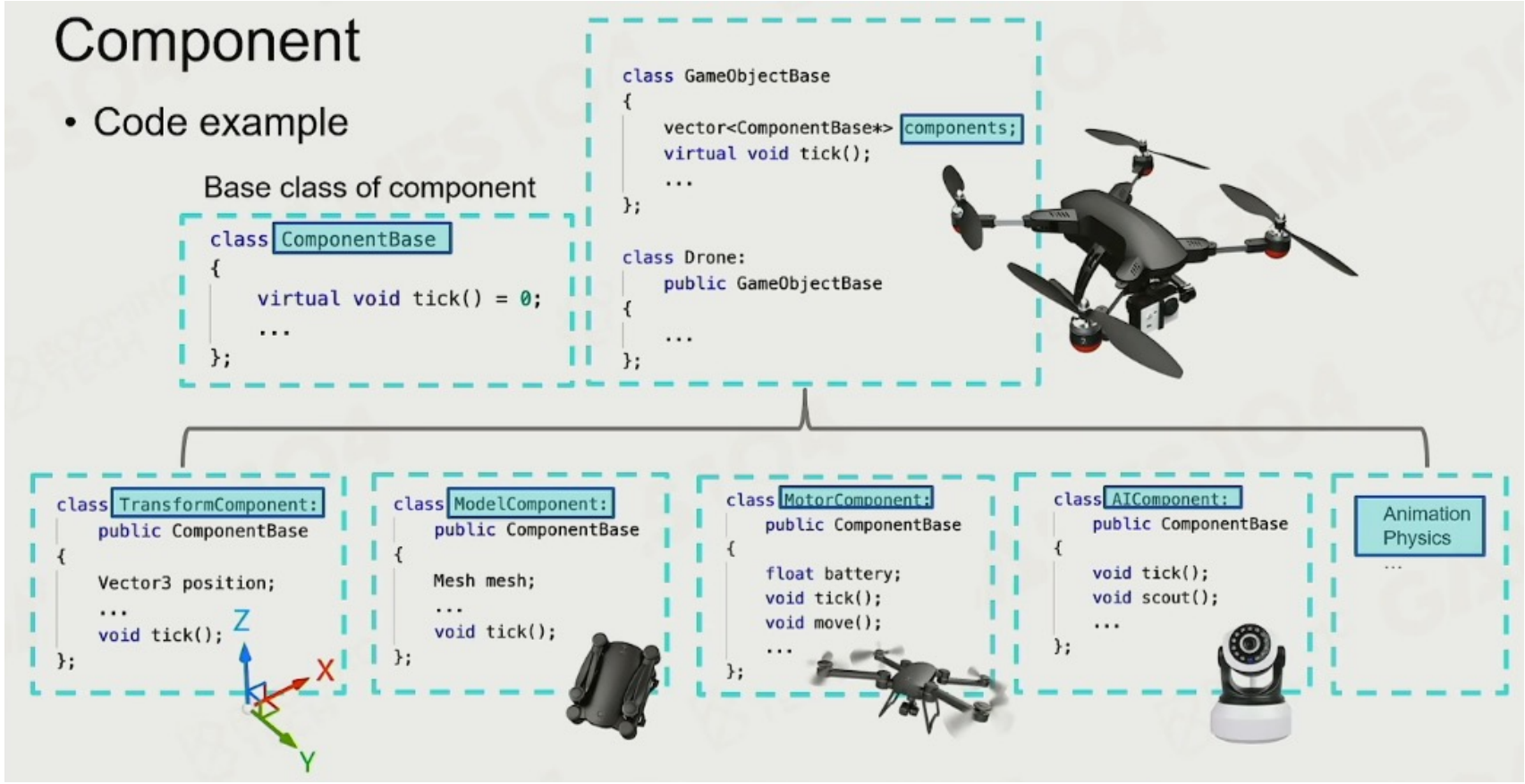
Flexible of coding languages



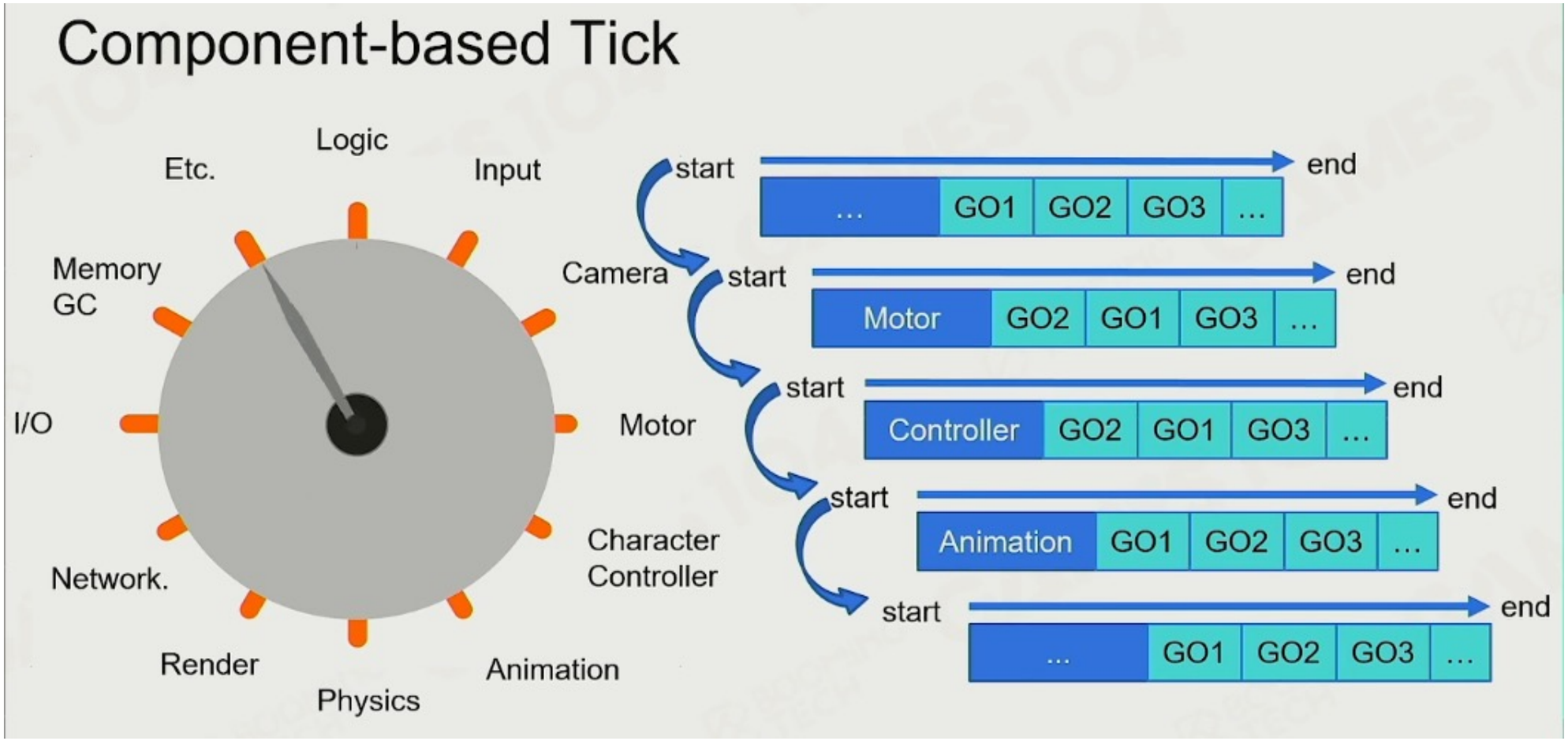
- Digital content creation (DCC

How to Build a Game World

- Game Object (GO)
  - Name, property, behavior.
  - Inheritance -> component base



Pipeline – batch process



Events

- Message sending and handling
- Decoupling event sending and handling

Soldier

Helicopter

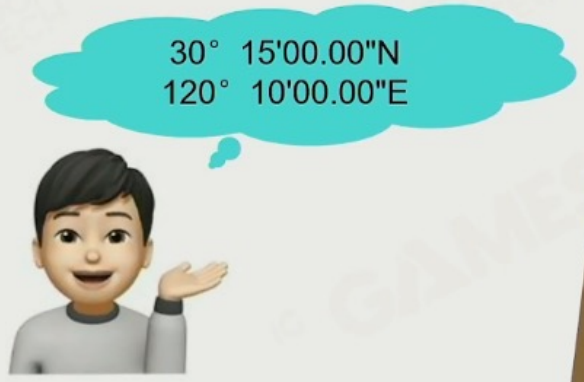
```
void Bomb:explode()
{
    ...
    sendExplodeEvent(go_id)
    ...
}
```

Tank

Stone

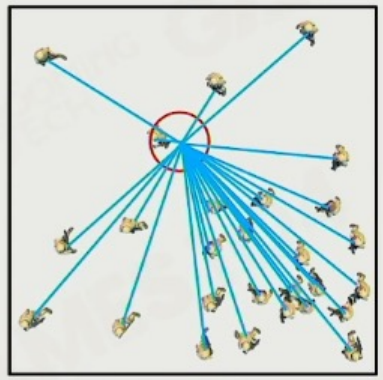
Scene Management

- Game objects are managed in a scene
- Game object query
  - By unique game object ID
  - By object position

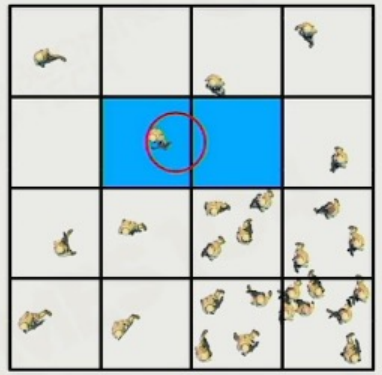


Scene Management

- Simple space segmentation



No division



Divided by grid

Scene Management

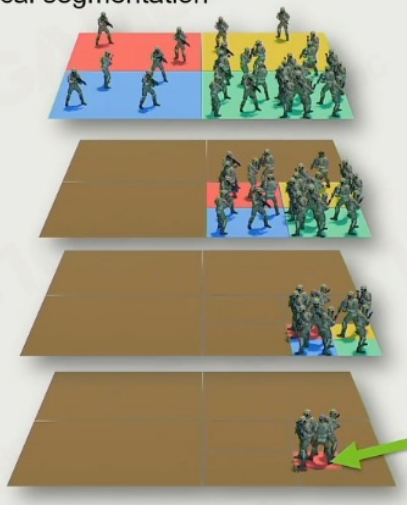
- Segmented space by object clusters
- Hierarchical segmentation

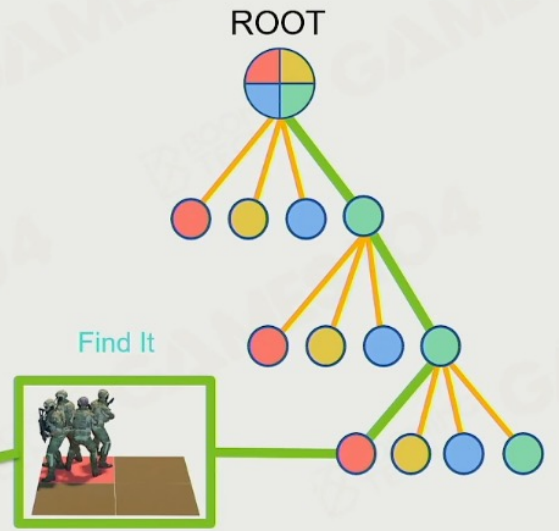
Divided by grid

Quadtree

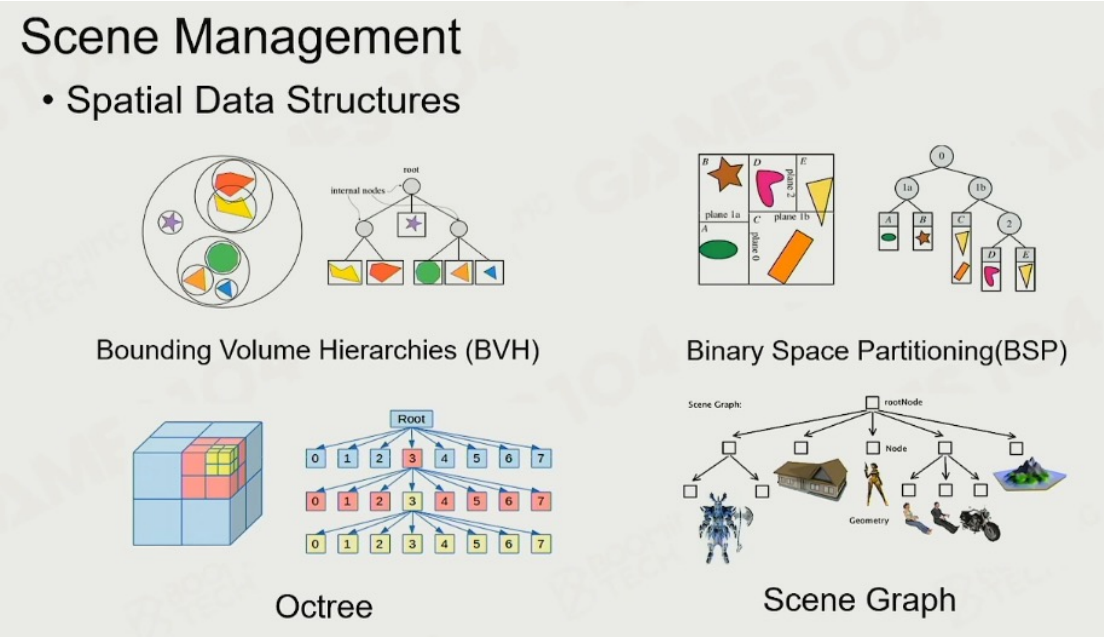
Scene Management

- Segmented space by object clusters
- Hierarchical segmentation

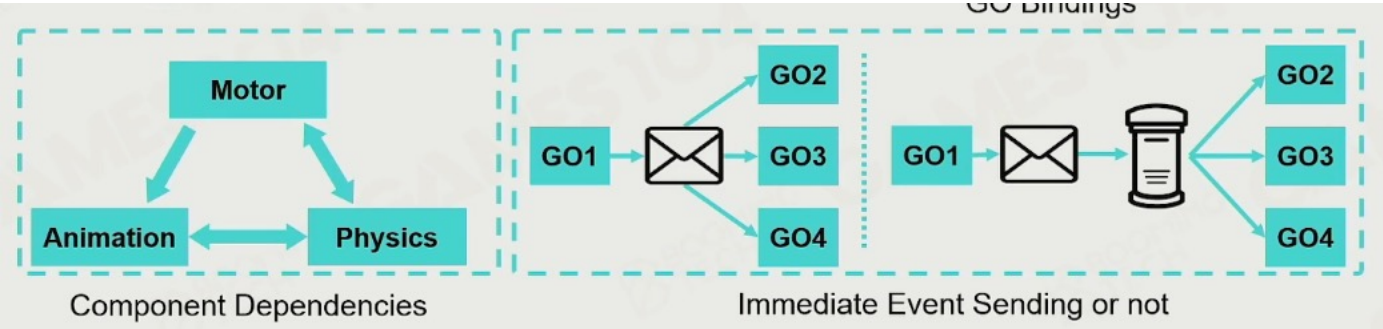




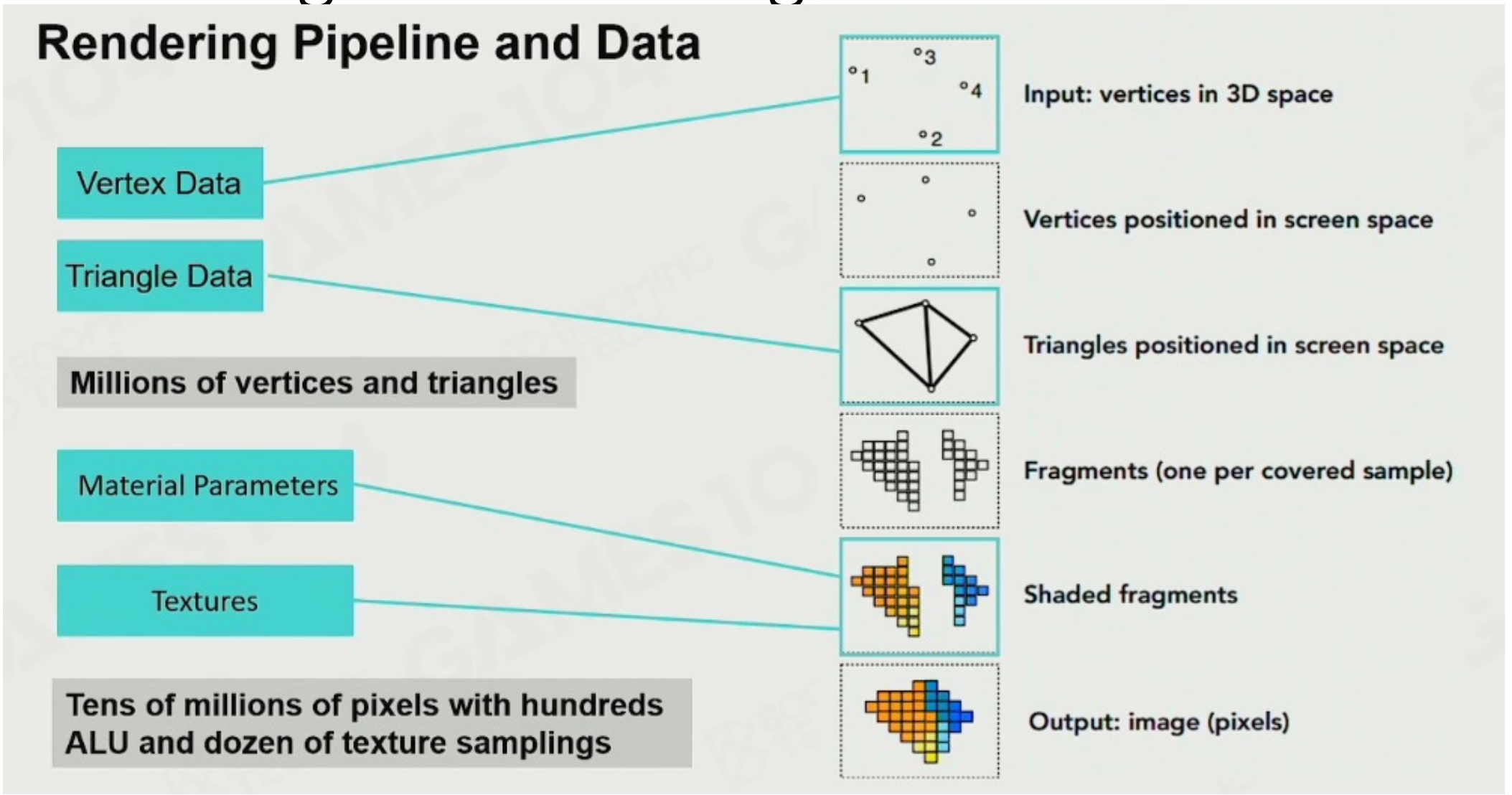
Bounding box



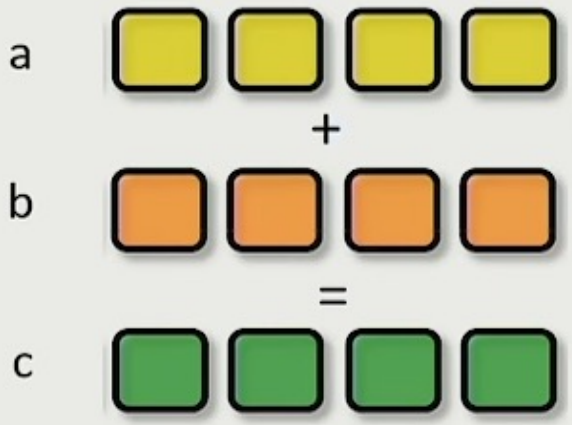
parallel

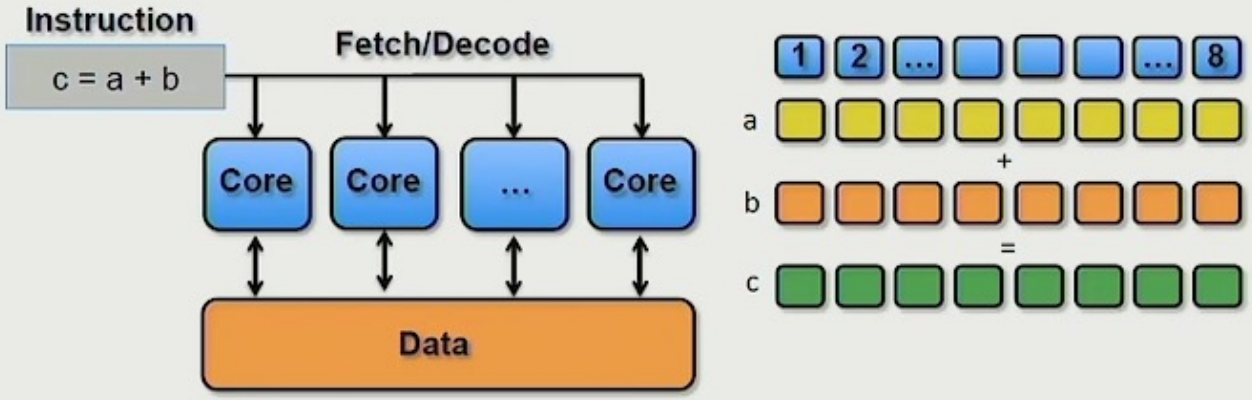


Rendering on Game Engine



SIMD and SIMT






```
SIMD_ADD c, a, b
```

```
SIMT_ADD c, a, b
```

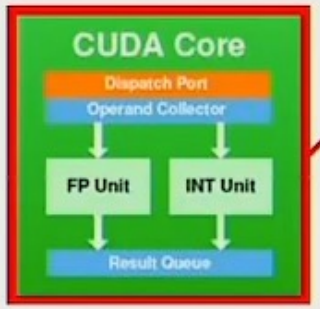
SIMD (Single Instruction Multiple Data)

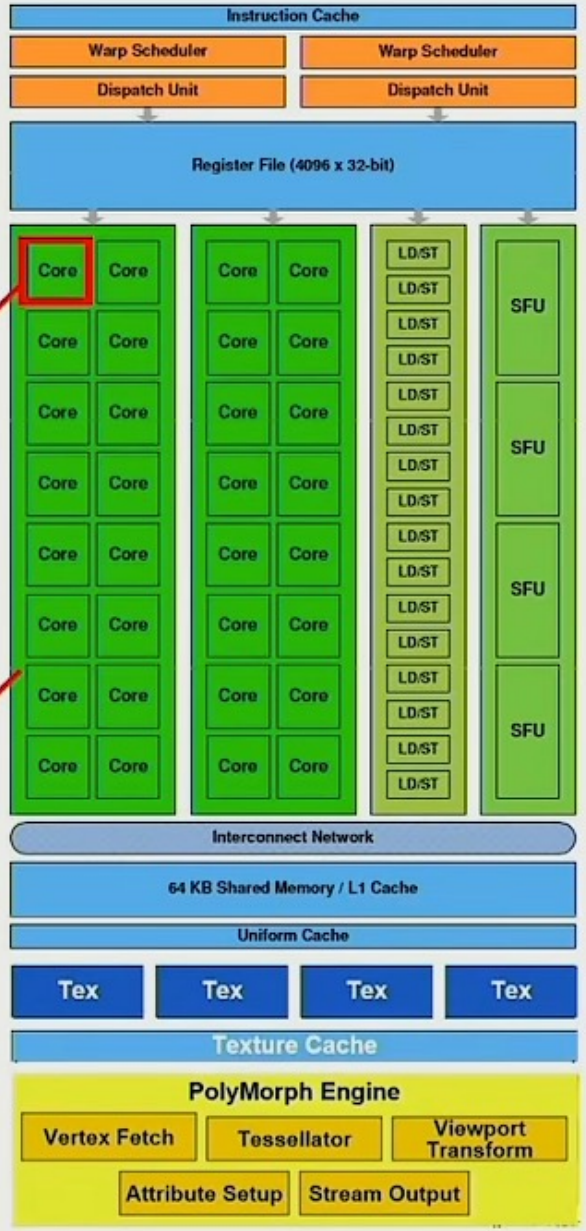
- Describes computers with multiple processing elements that perform the same operation on multiple data points simultaneously

SIMT (Single Instruction Multiple Threads)

- An execution model used in parallel computing where single instruction, multiple data (SIMD) is combined with multithreading

GPU Architecture





GPC (Graphics Processing Cluster)
A dedicated hardware block for computing, rasterization, shading, and texturing

SM (Streaming Multiprocessor)
Part of the GPU that runs CUDA kernels

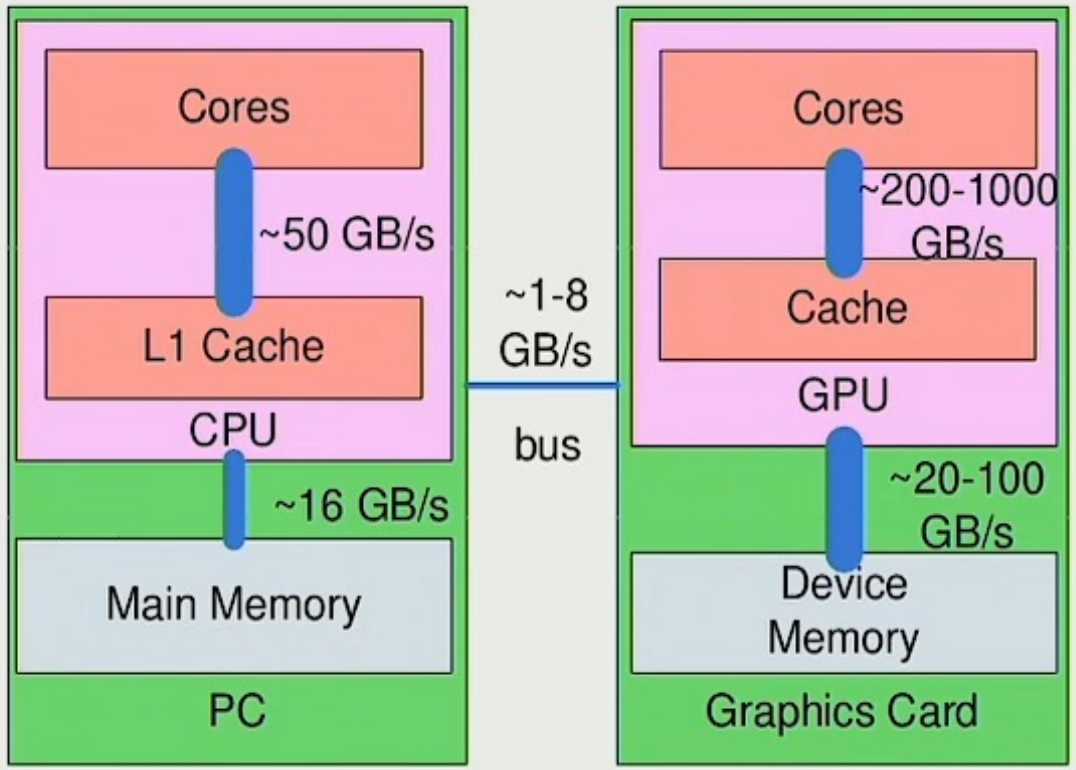
Texture Units
A texture processing unit, that can fetch and filter a texture

CUDA Core
Parallel processor that allow data to be worked on simultaneously by different processors

Warp
A collection of threads

e.g. Fermi Architecture

Data Flow from CPU to GPU



- **CPU and Main Memory**
  - Data Load / Unload
  - Data Preparation
- **CPU to GPU**
  - High Latency
  - Limited Bandwidth
- **GPU and Video Memory**
  - High Performance Parallel Rendering

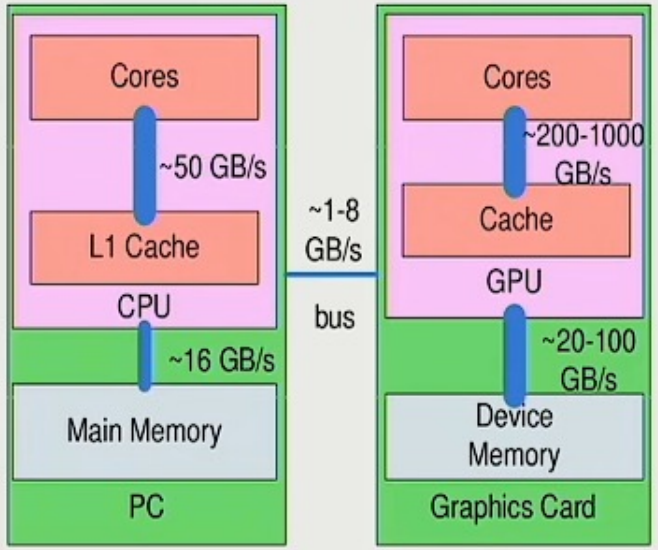
Tips

Always minimize data transfer between CPU and GPU when possible

Better not to send back from GPU to CPU

Be Aware of Cache Efficiency

- Take full advantage of hardware parallel computing
- Try to avoid the von Neumann bottleneck



CPU Cache Access Latencies in Clock cycles

Mainmemory		167
L3 Cache		38
L2 Cache		11
L1 Cache		4

GPU L2 Cache Access Latencies (measu

Amphere L2	100ns
RDNA L2	20ns

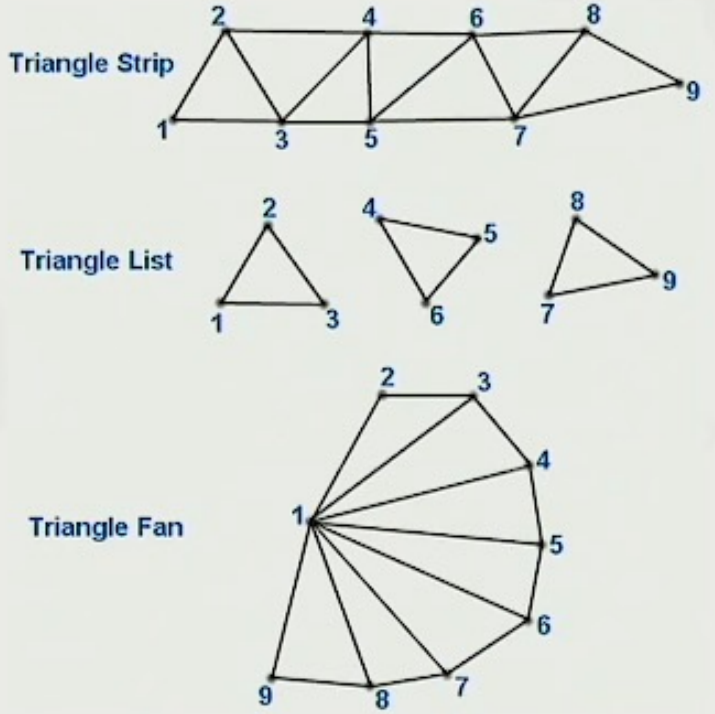
NVIDIA AM

GPU Bounds and Performance

Application performance is limited by:

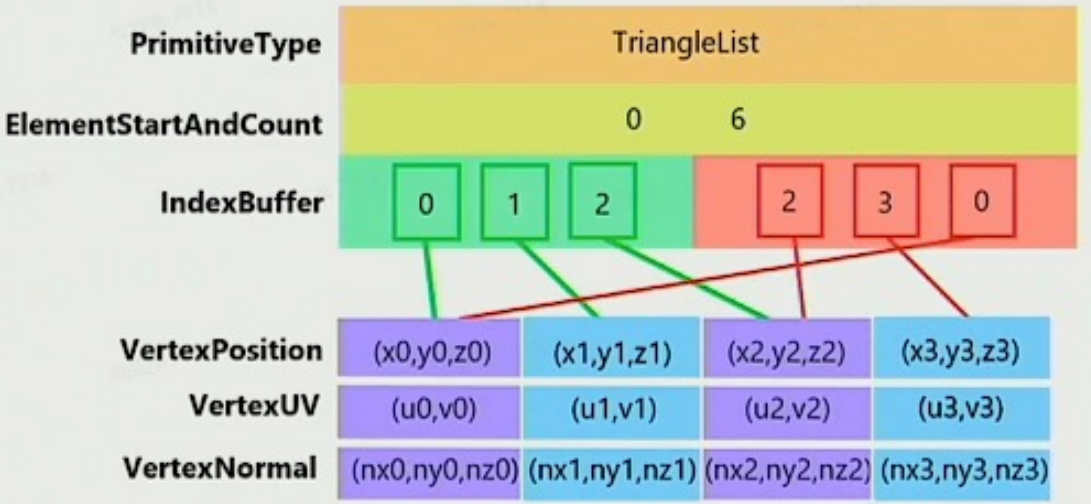
- **Memory Bounds**
- **ALU Bounds**
- **TMU (Texture Mapping Unit) Bound**
- **BW (Bandwidth) Bound**

Vertex and Index Buffer

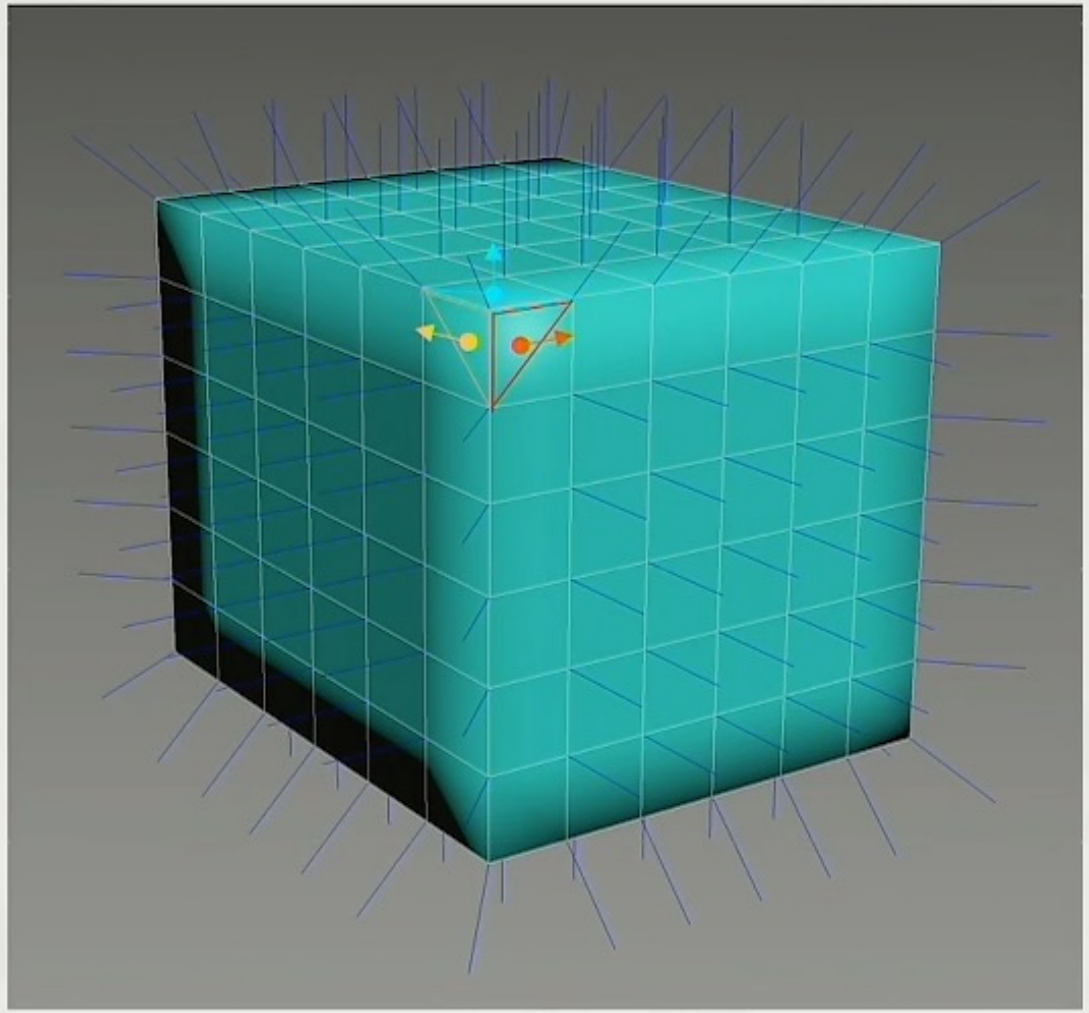


- **Vertex Data**
  - Vertex declaration
  - Vertex buffer

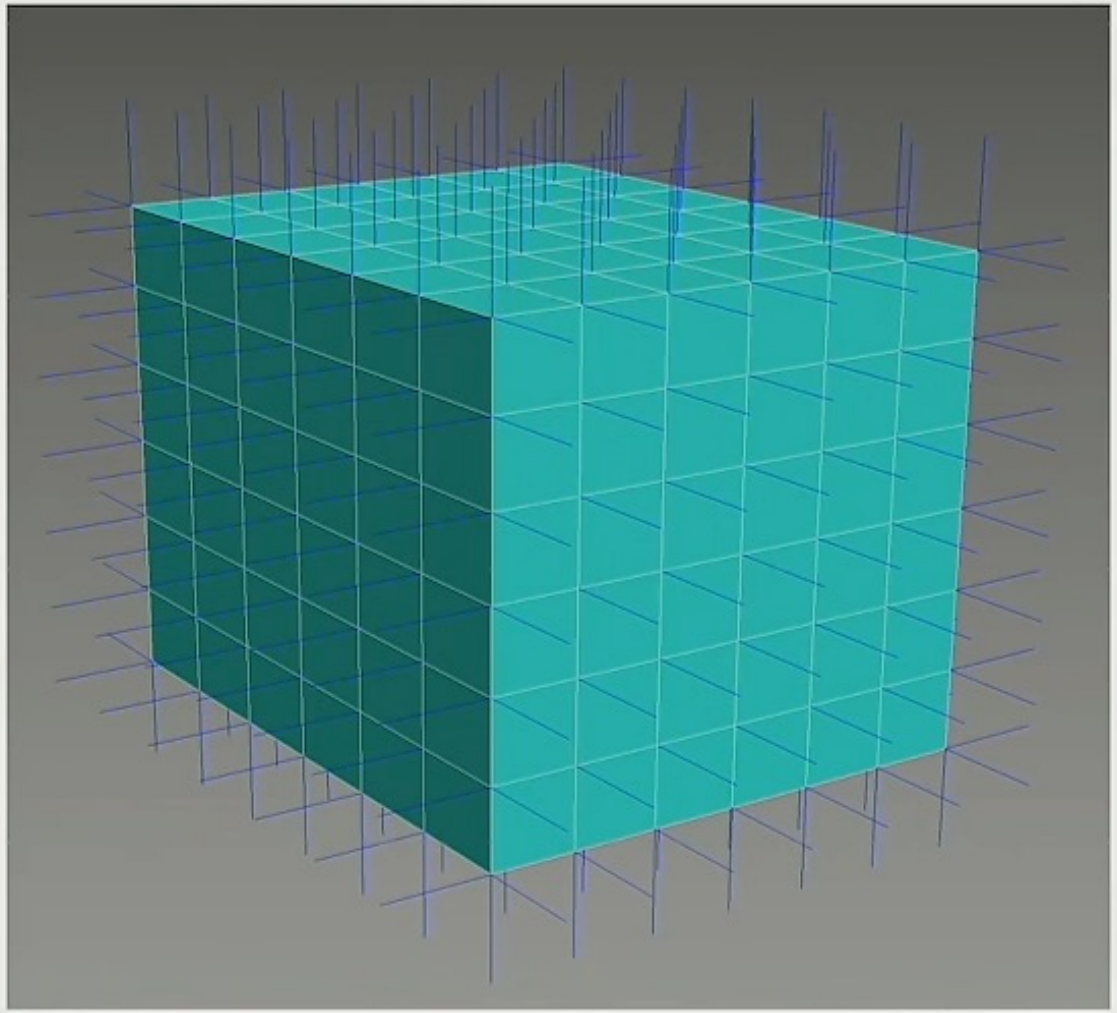
- **Index Data**
  - Index declaration
  - Index buffer



Why We Need Per-Vertex Normal



Interpolate vertex normal by triangle normal



Per-Vertex normals necessary

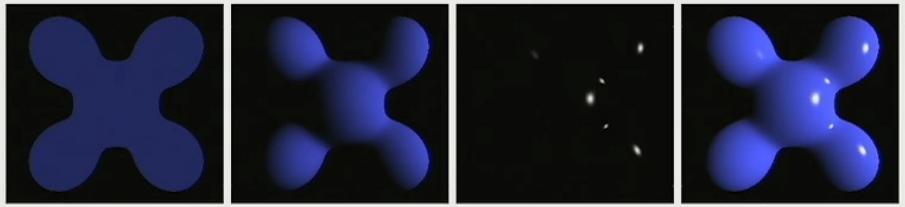
Same vertex different normal for sharp angle.

Materials

Base | Smooth metal | Glossy paint | Rough stone | Transparent glass

Determine the appearance of objects, and how objects interact with light

Famous Material Models

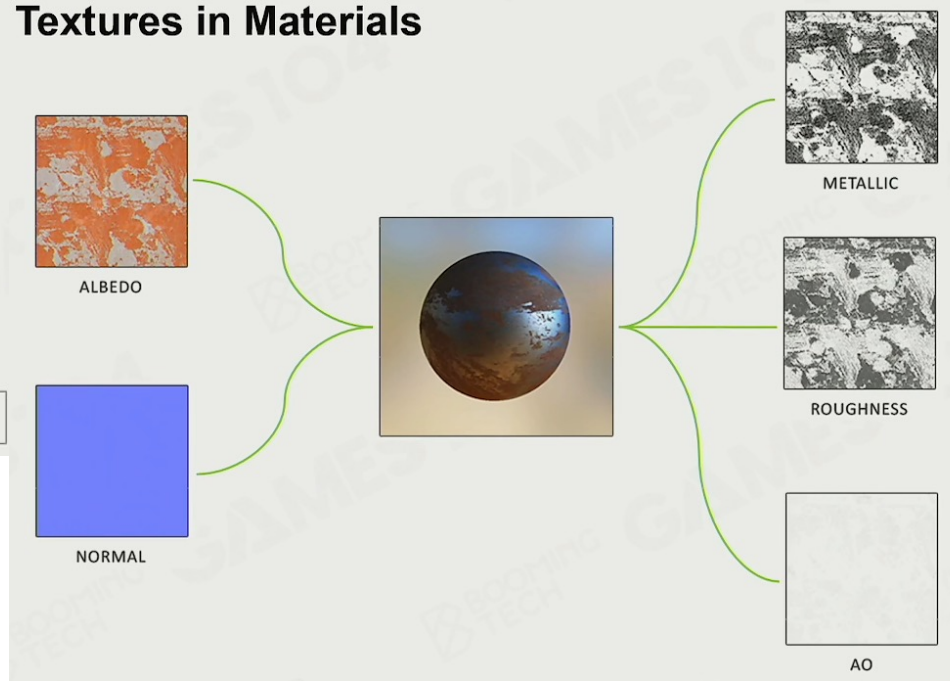


Phong Model

PBR Model - Physically based rendering

Subsurface Material - Burley SubSurface Profile

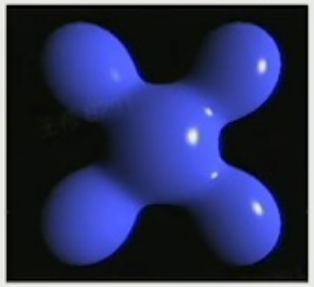
Textures in Materials



Variety of Shaders

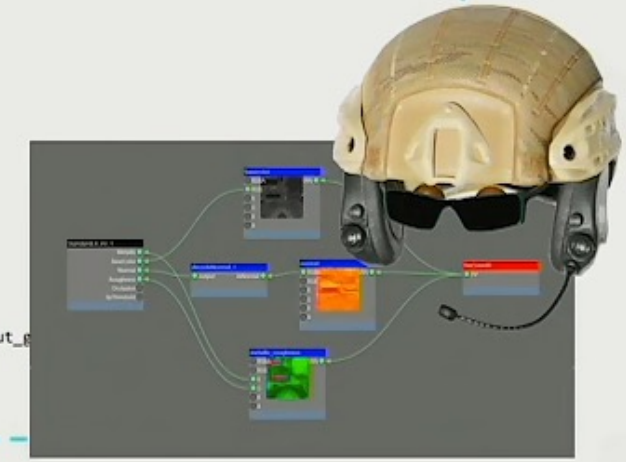
Fix Function Shading Shaders

```
float4 PSMain(PixelInput input) : SV_TARGET
{
    float3 world_normal = normalize(input.world_normal);

    float3 world_view_dir = normalize(world_space_camera_pos - input.world_pos);

    float3 world_light_reflection_dir = normalize(reflect(-world_light_dir, world_normal));

    float3 ambient = ambient_color * material.ambient;

    float3 diffuse = max(0, dot(world_normal, world_light_dir)) *
                     diffuse_color * material.diffuse;

    float3 specular = pow(max(0, dot(world_light_reflection_dir, world_view_dir)), shininess) *
                      specular_color * material.specular;

    float3 emissive = material.emissive;

    return float4(ambient + diffuse + specular  + emissive, 1.0f);
}
```

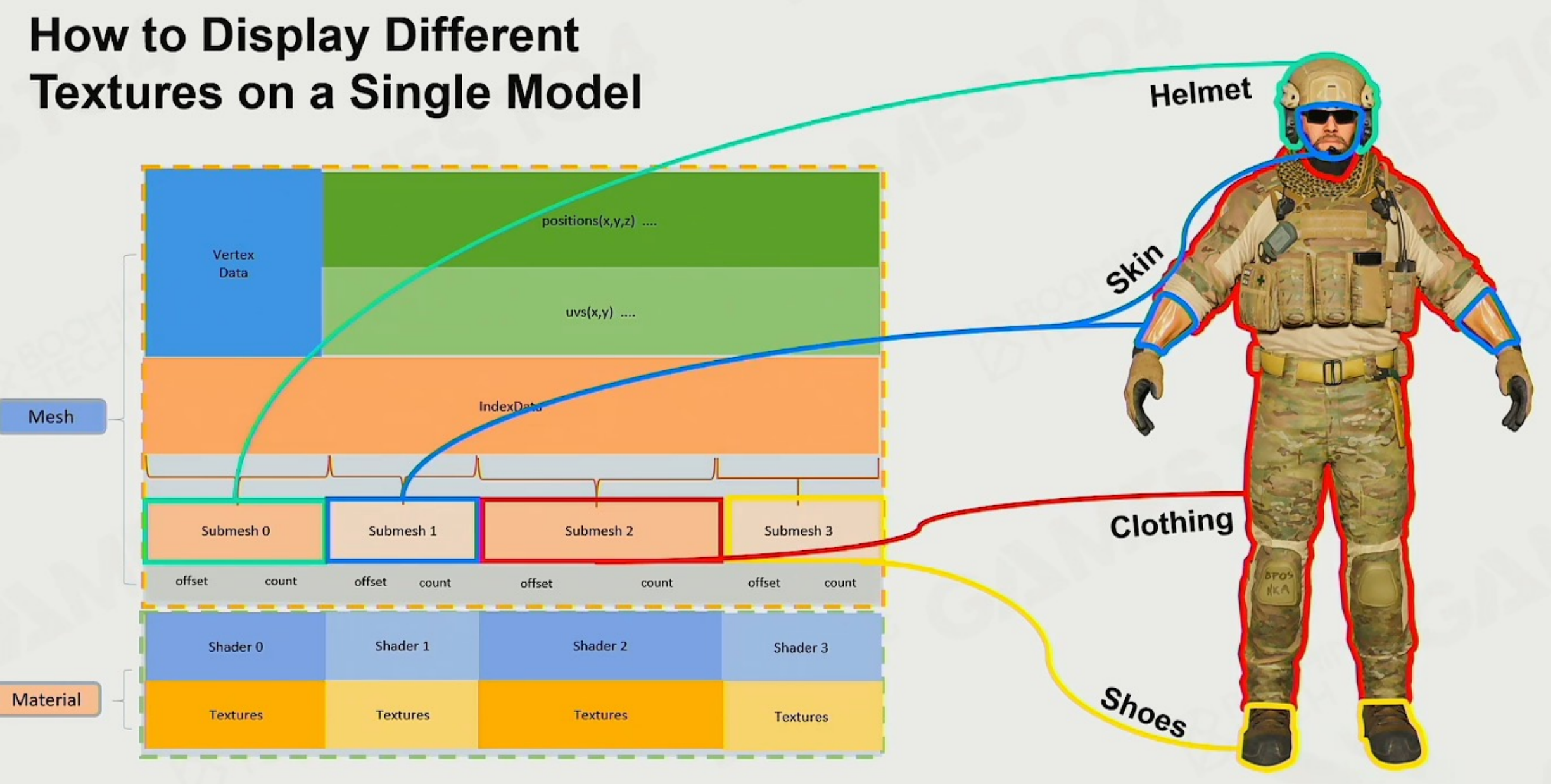


Custom Shaders

```
104  PixelOutGbuffer PS_Entry_deferred(PixelInput input)
105  {
106      float3 T = input.world_tangent.xyz;
107      float3 N = normalize(input.world_geo_normal);
108      float3 B = cross(input.world_geo_normal, input.world_tangent.xyz)
109          * input.world_tangent.w;
110
111      T -= dot(T, N) * N;
112      T = normalize(T);
113      B -= dot(B, N) * N + dot(B, T) * T;
114      B = normalize(B);
115
116      float3x3 TBN; TBN[0] = T; TBN[1] = B; TBN[2] = N;
117
118      GBufferData gbuffer_data;
119      initializeGBufferData(gbuffer_data);
120
121      //albedo
122      float4 albedo_opacity_value = CHAOS_SAMPLE_TEX2D(albedo_opacity_map, input.tex0);
123      gbuffer_data.albedo = albedo_opacity_value.rgb;
124
125      //normal
126      float3 normal_value = decodeNormalFromNormalMapValue(normal_map.rgba).rgb;
127      gbuffer_data.world_normal =  normalize(mul(normal_value.rgb, TBN));
128
129      //specular
130      float4 specular_glossiness_value = CHAOS_SAMPLE_TEX2D(specular_glossiness_map, input.tex0);
131      gbuffer_data.reflectance =  specular_glossiness_value.rgb;
132
133      //smoothness
134      gbuffer_data.smoothness =  specular_glossiness_value.r;
135
136      //ao
137      gbuffer_data.ao =  occlusion;
138
139      //opacity
140      float albedo_opacity_value = albedo_opacity_value.a;
141
142      float  alpha_clip_value = alpha_clip;
143
144      clip(albedo_opacity_value - alpha_clip_value);
145
146      PixelOutGbuffer out_gbuffer = (PixelOutGbuffer)0;
147      EncodeGBuffer(gbuffer_data, out_gbuffer.GBufferA, out_gbuffer.GBufferB, out_g
148
149      return out_gbuffer;
150  }
```

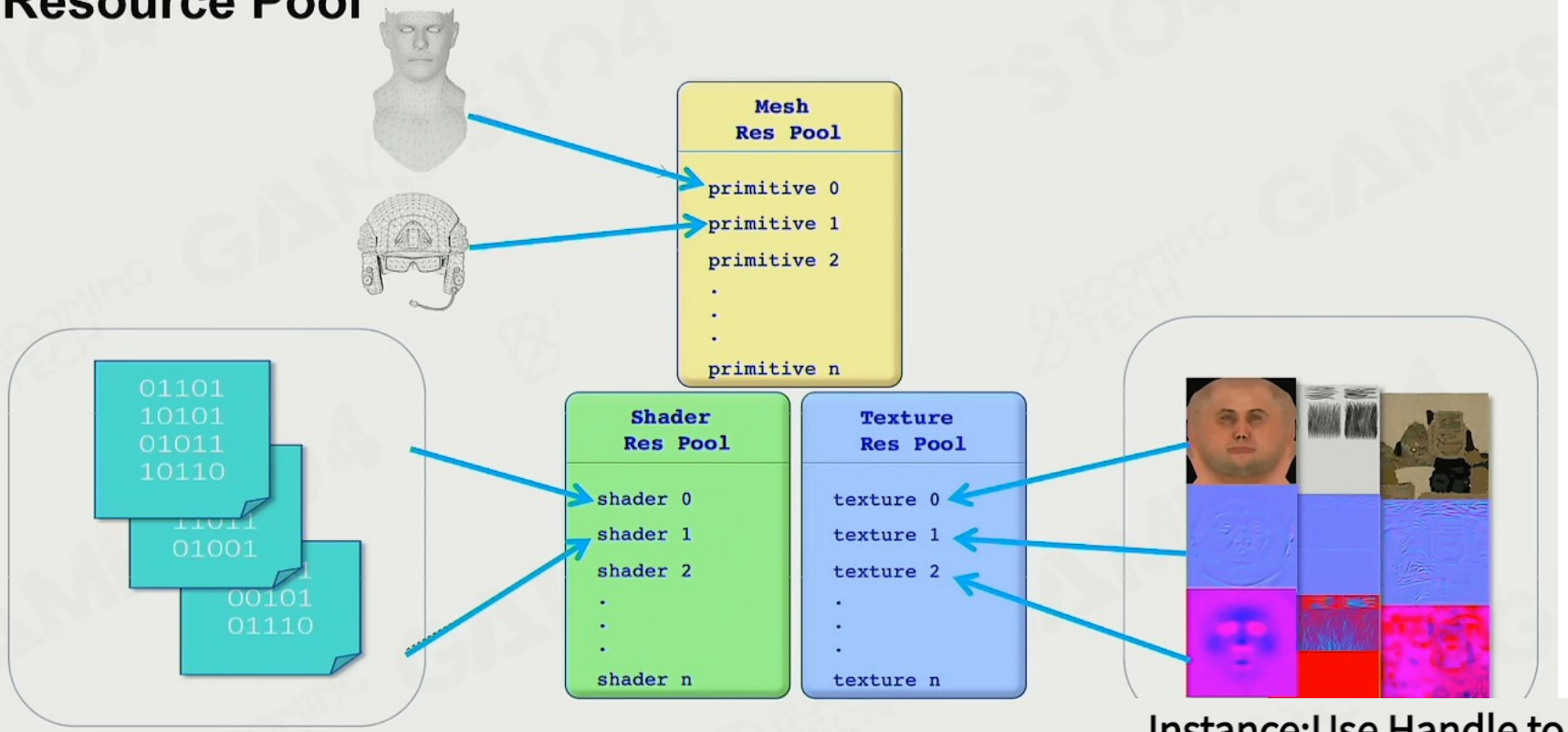


Shader code: source and also data.

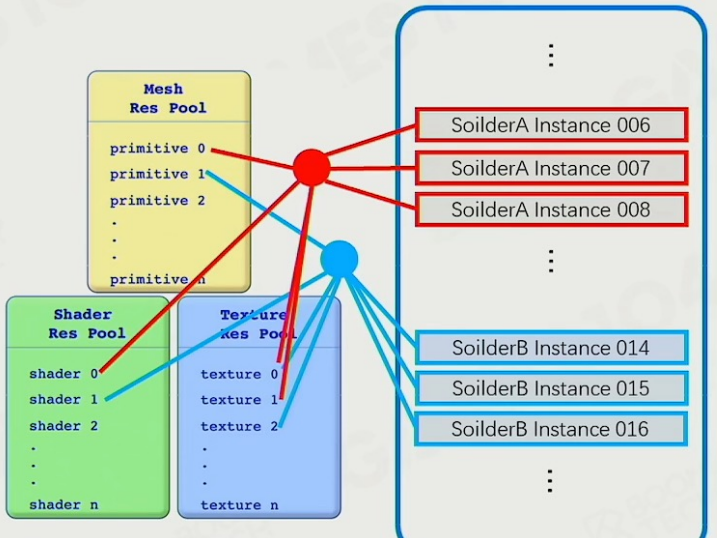

How to Display Different Textures on a Single Model
Helmet
Skin
Clothing
Shoes
Vertex Data
positions(x,y,z) ....
uvs(x,y) ....
IndexData
Mesh
Submesh 0
Submesh 1
Submesh 2
Submesh 3
offset
count
Shader 0
Shader 1
Shader 2
Shader 3
Material
Textures

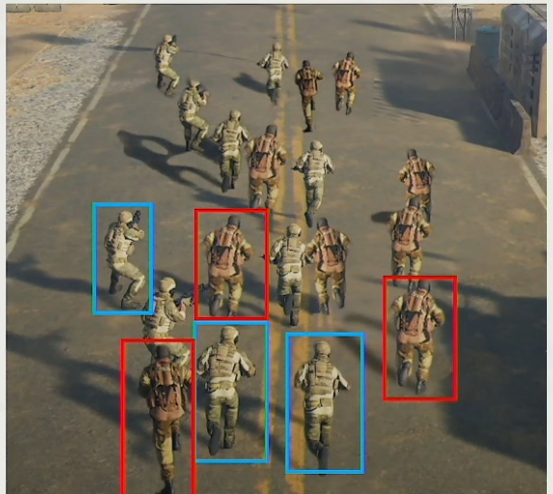
submesh

Resource Pool



Instance:Use Handle to Reuse Resources





Sort by Material

```
Initialize Resource Pools
Load Resources

Sort all Submeshes by Materials

for each Materials
    Update Parameters
    Update Textures
    Update Shader
    Update VertexBuffer
    Update IndexBuffer
    for each Submeshes
        Draw Primitive
    end
end
```

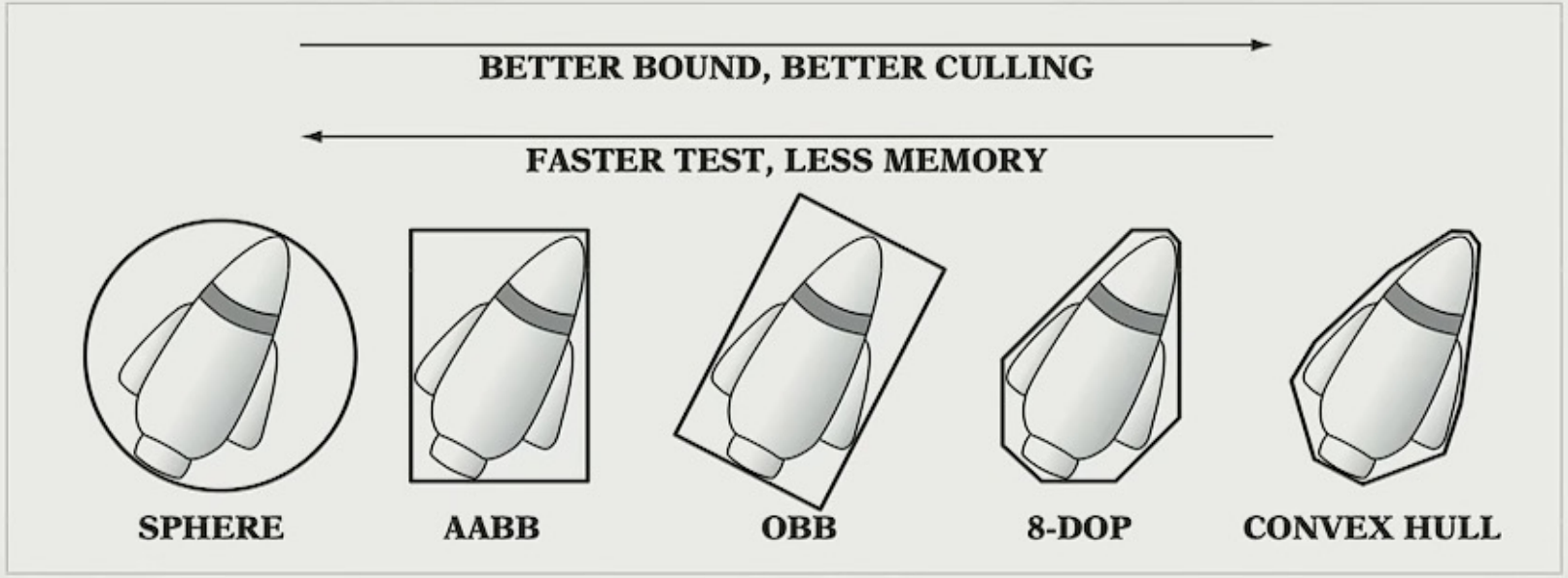
GPU Batch Rendering

```
struct batchData
{
    SubmeshHandle m_submesh_handle;
    MaterialHandle m_material_handle;
    std::vector<PerInstanceData> m_per_instance_data;
    unsigned int m_instance_count;
};

Initialize Resource Pools
Load Resources

Collect batchData with same submesh and material

for each BatchData
    Update Parameters
    Update Textures
    Update Shader
    Update VertexBuffer
    Update IndexBuffer
        Draw Instance
end
```

Q : What if group rendering all instances with identical submeshes and materials together?

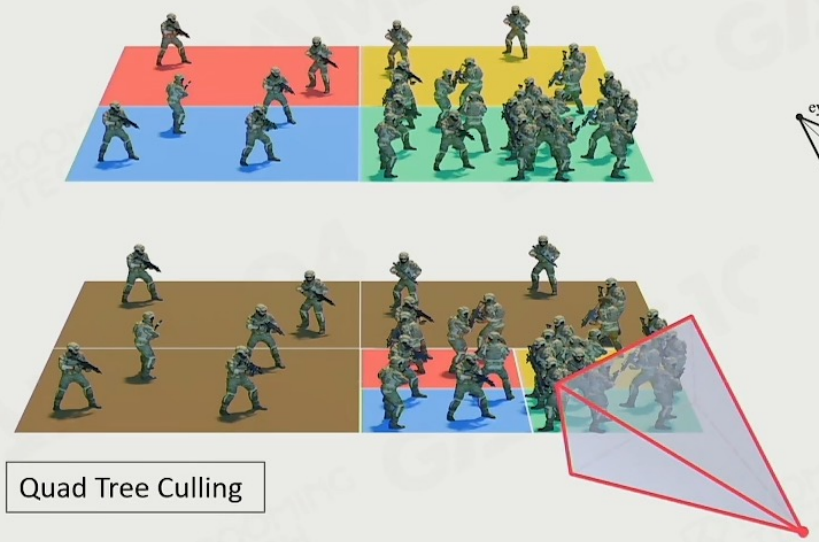
Visibility Culling 可见性裁剪

Using the Simplest Bound to Create Culling

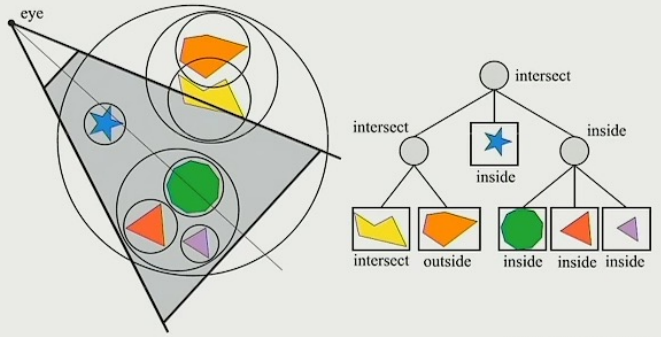


- Inexpensive intersection tests
- Tight fitting
- Inexpensive to compute
- Easy to rotate and transform
- Use little memory

Hierarchical View Frustum Culling



Quad Tree Culling

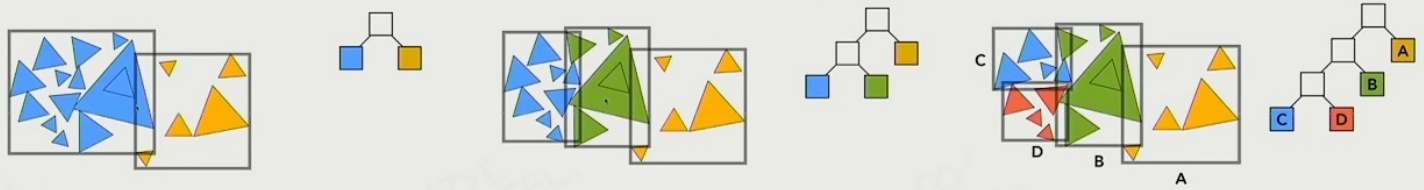


BVH (Bounding Volume Hierarchy) Culling

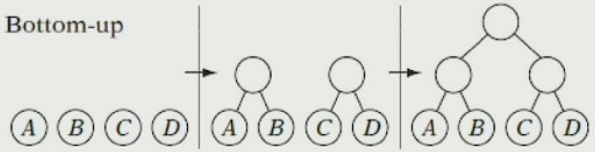
BVH

Construction and Insertion of BVH in Game Engine

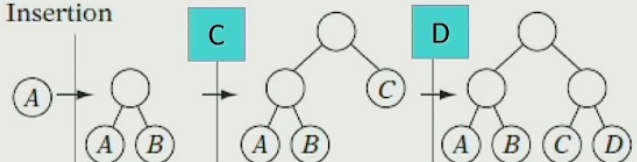
Top-down



Bottom-up



Incremental tree-insertion



PVS (Potential Visibility Set)

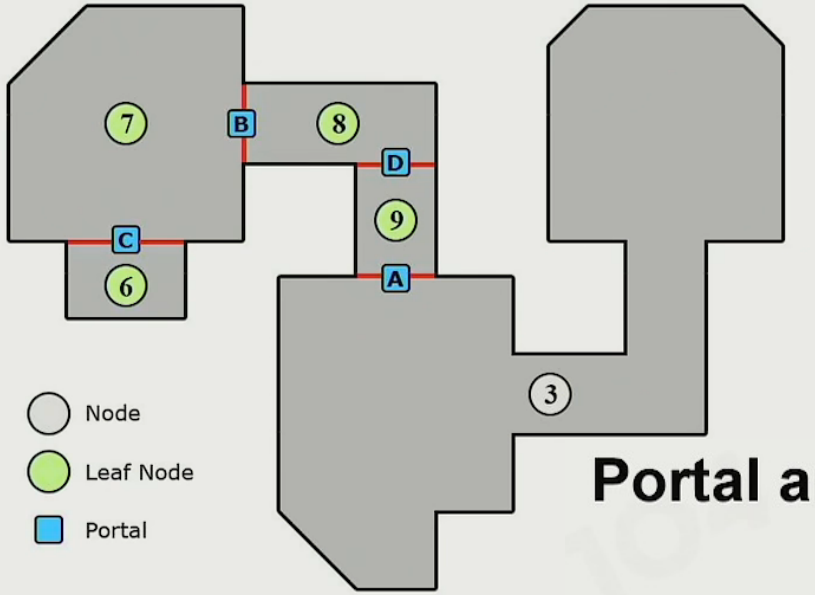
Idea used in Resource loading

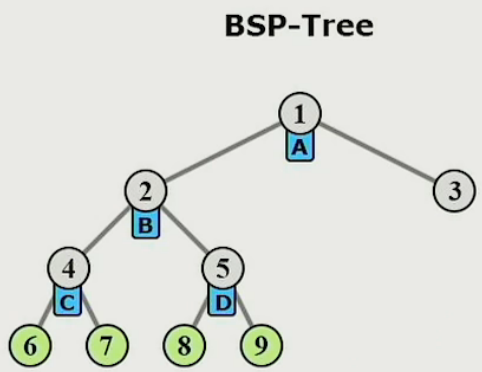
Green box:

The area to determine the potential visibility where you need.

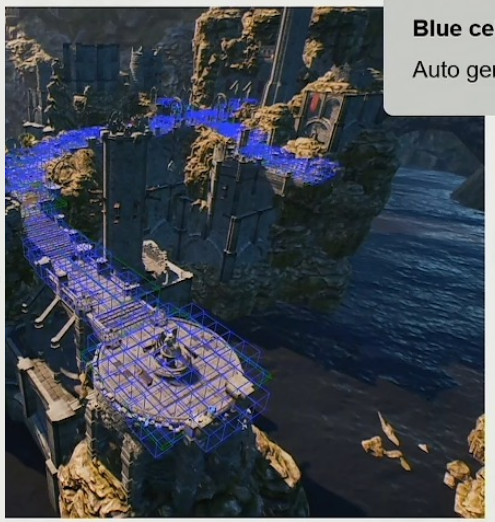
Blue cells:

Auto generated smaller regions of each green box.





Portal and PVS Data



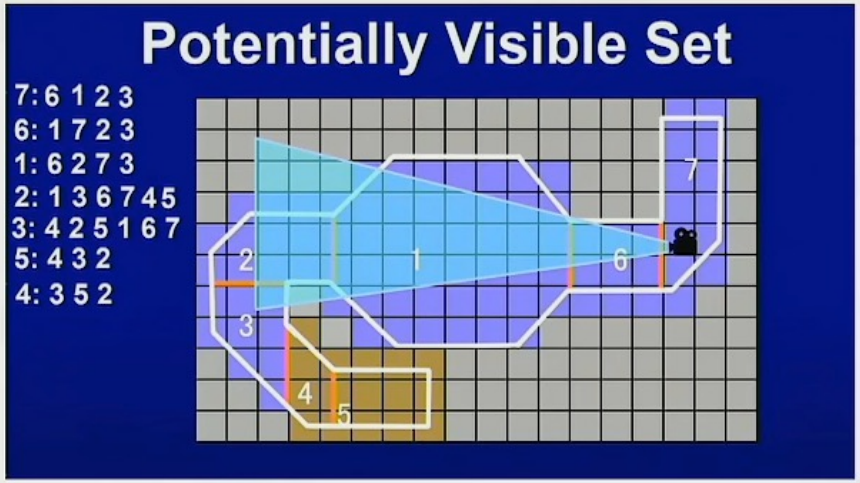
```
for each GreenBoxs
    for each BlueCells
        do ray casting between box and cell
    end
end
```

Pros

- Much faster than BSP / Octree
- More flexible and compatible
- Preload resources by PVS

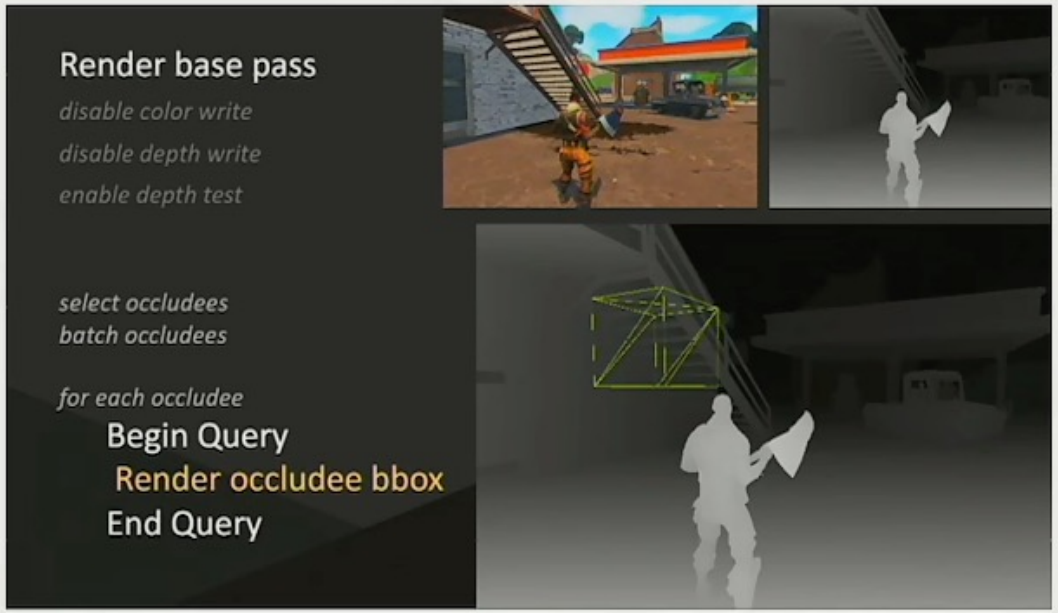
Generate PVS data from portal:

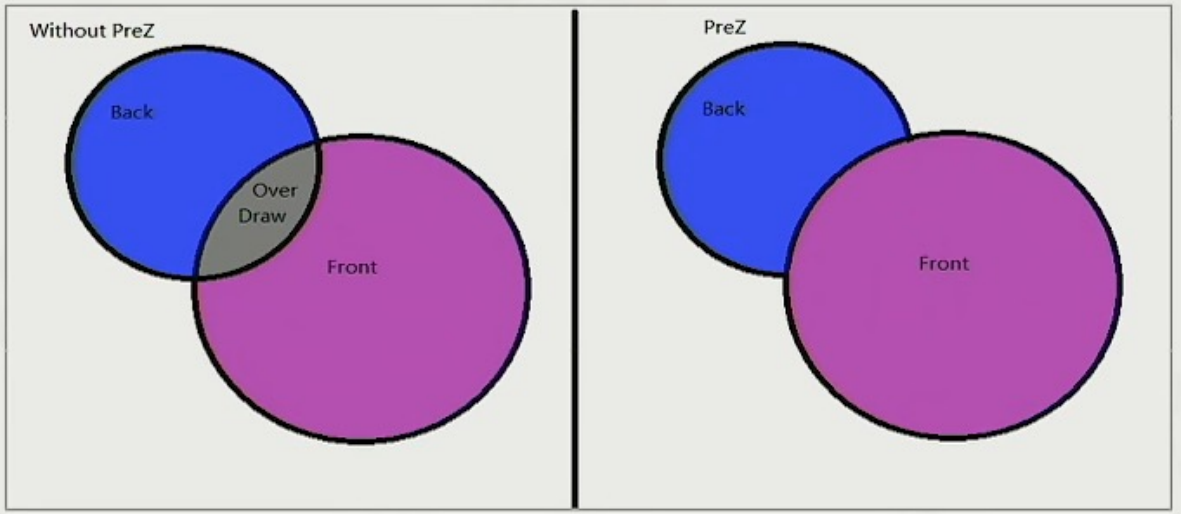
```
for each portals
    getSamplingPoints();
    for each portal faces
        for each leaf
            do ray casting between portal face and leaf
        end
    end
end
```

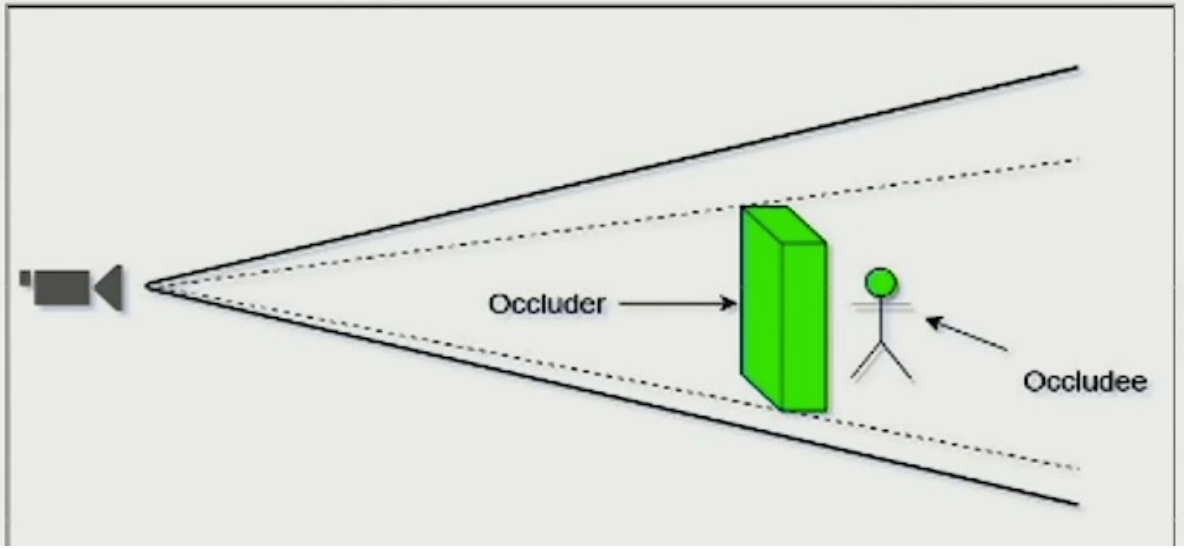


Determine potentially visible leaf nodes immediately from portal

GPU Culling





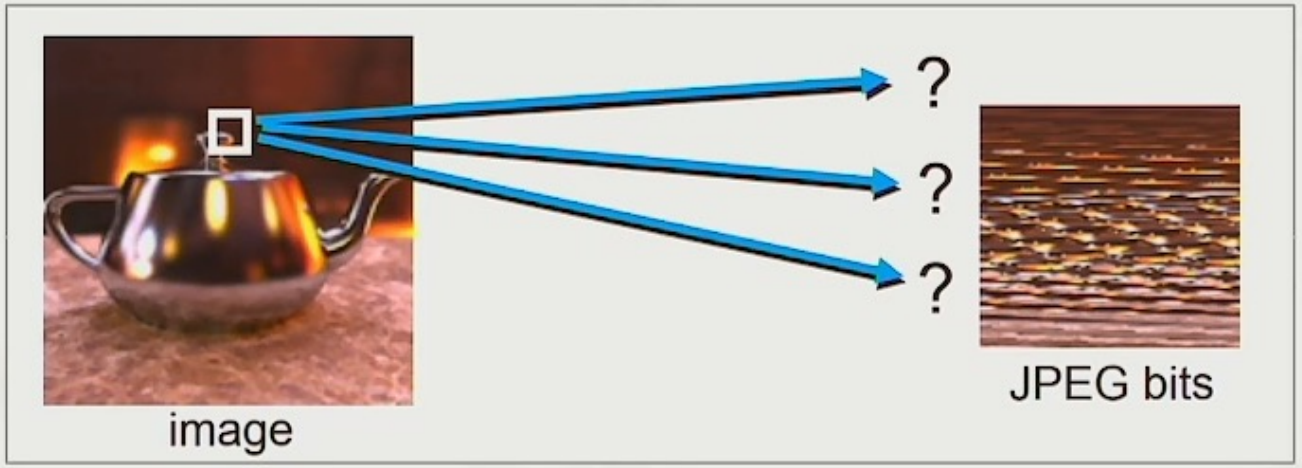


Seems like Z-buffer

Texture Compression

- **Traditional image compression like JPG and PNG**
  - Good compression rates
  - Image quality
  - Designed to compress or decompress an entire image

- **In game texture compression**
  - Decoding speed
  - Random access
  - Compression rate and visual quality
  - Encoding speed

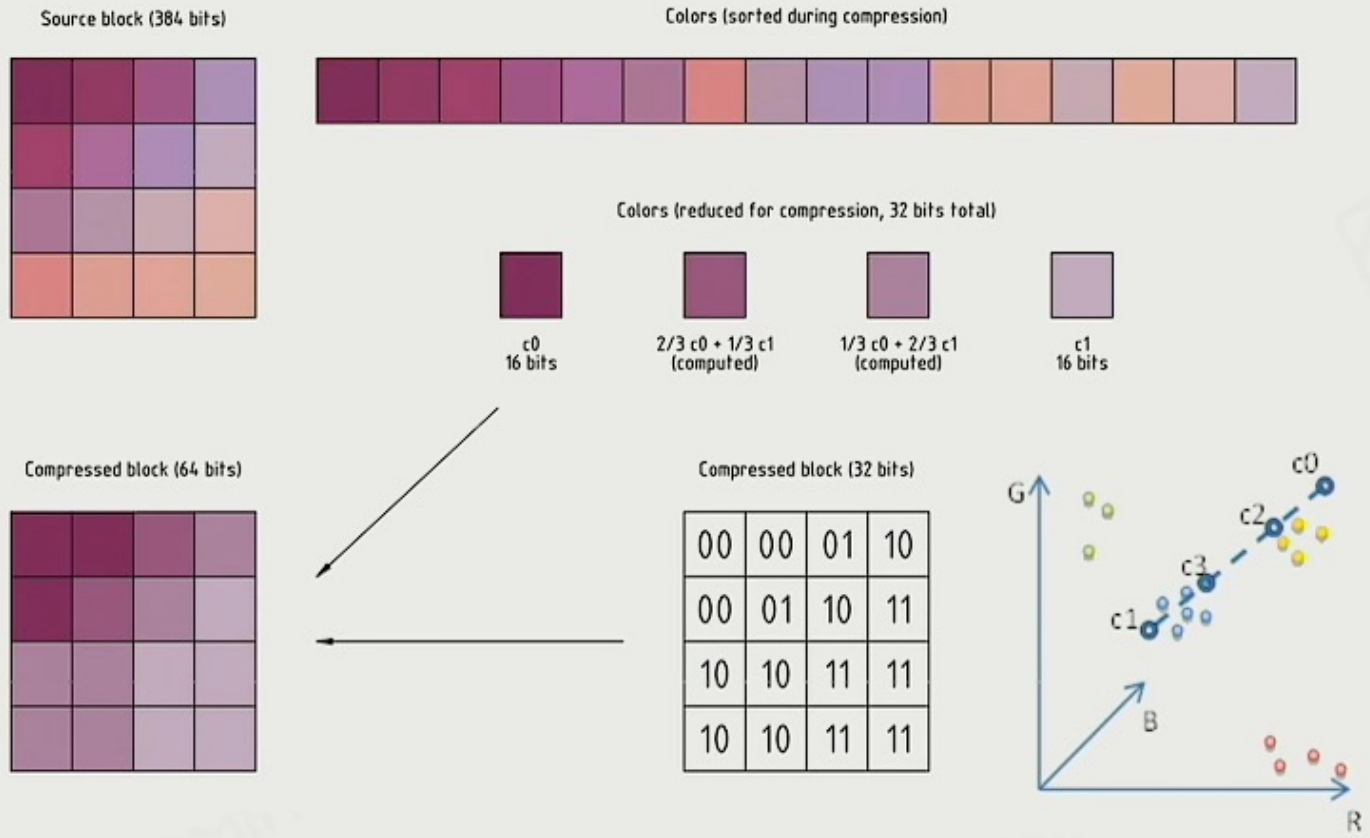


Sample JPEG format texture

Block Compression

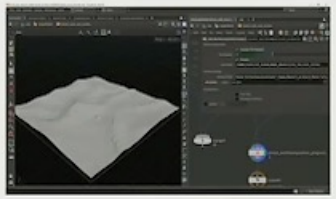
Common block-based compression format

- On PC, BC7 (modern) or DXTC (old) formats
- On mobile, ASTC (modern) or ETC / PVRTC (old) formats



Modeling

Comparison of Authoring Methods

	Polymodeling	Sculpting	Scanning	Procedural modeling
Sample				
Advantage	Flexible	Creative	Realistic	Intelligent
Disadvantage	Heavy workload	Large volume of data	Large volume of data	Hard to achieve

Maya, max, blender zbrush Houdini, unreal

Cluster-Based Mesh Pipeline

GPU-Driven Rendering Pipeline (2015)

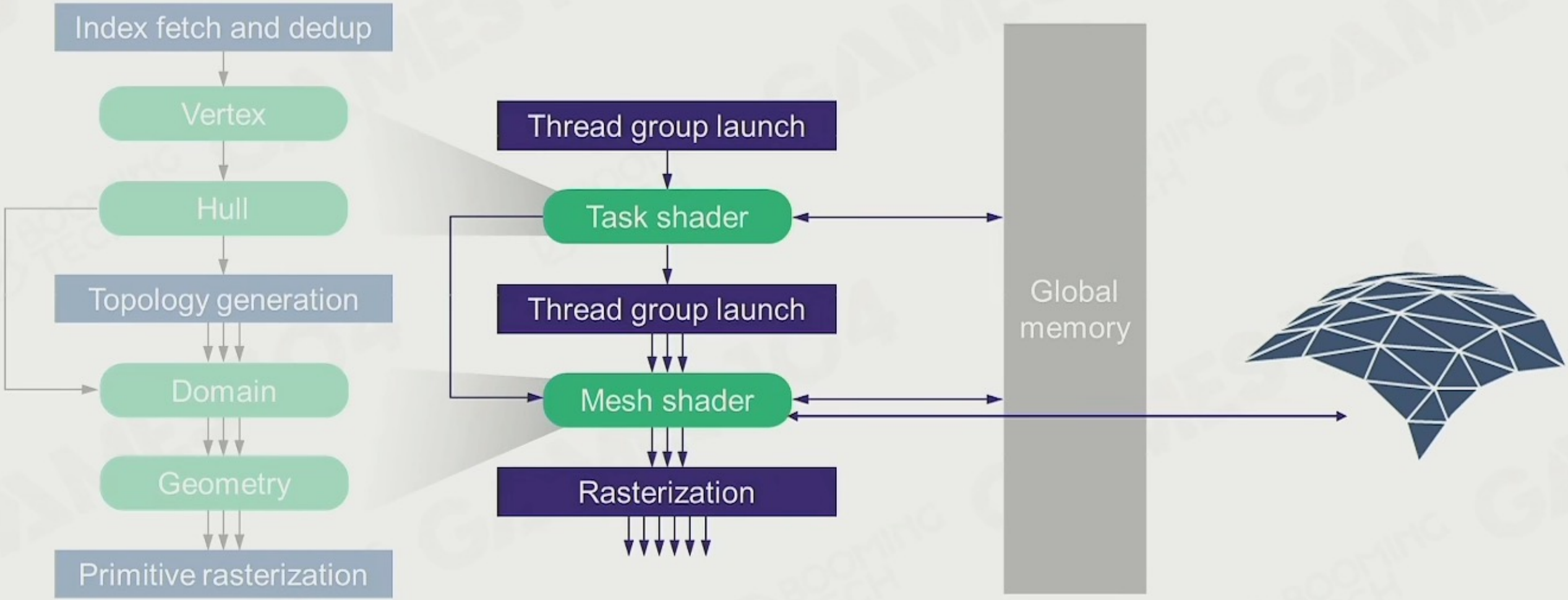
- Mesh Cluster Rendering
  - Arbitrary number of meshes in single drawcall
  - GPU-culled by cluster bounds
  - Cluster depth sorting

Geometry Rendering Pipeline Architecture (2021)

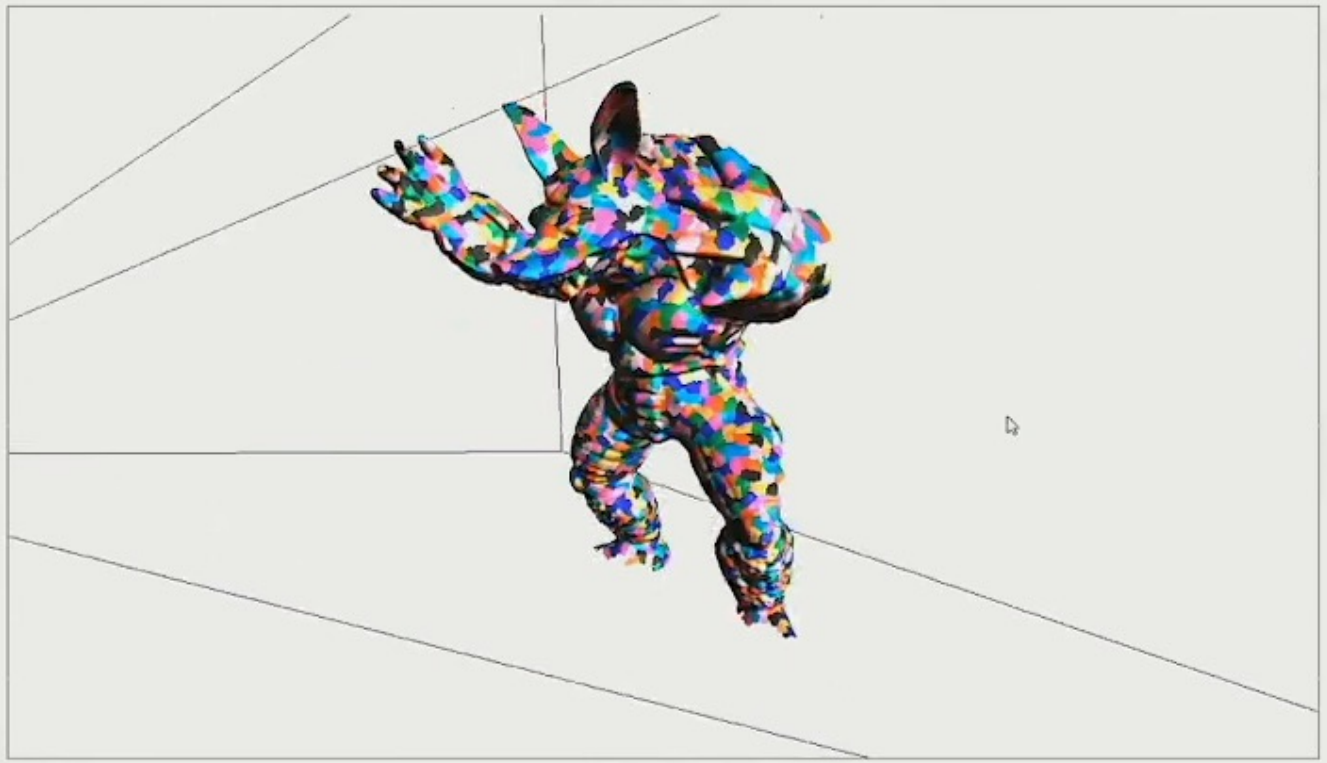
- Rendering primitives are divided as:
  - Batch: a single API draw (drawIndirect / drawIndexIndirect), composed of many Surfs
  - Surf: submeshes based on materials, composed of many Clusters
  - Cluster: 64 triangles strip

New trend

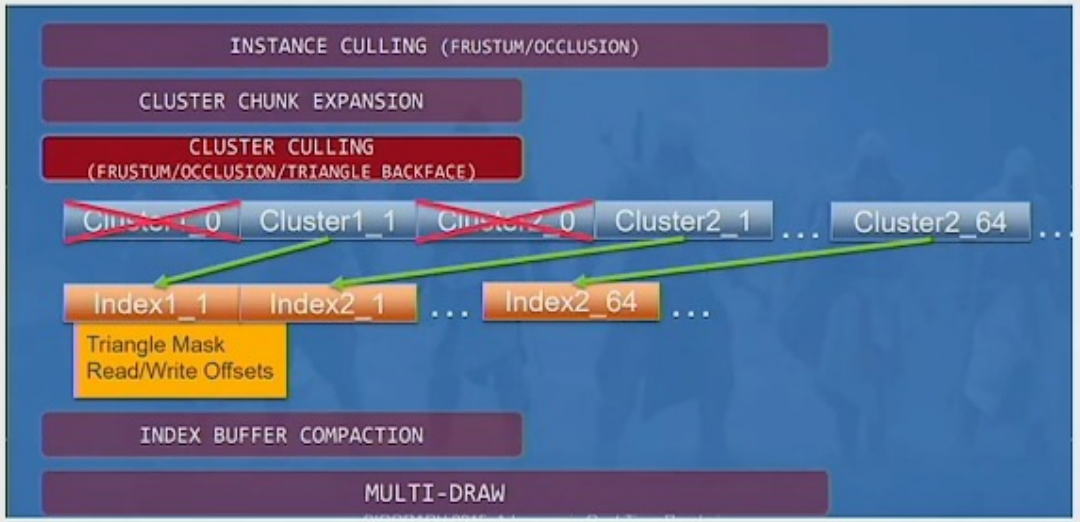
Programmable Mesh Pipeline



GPU Culling in Clustered-Based Mesh



350k triangles to 2791 clusters



GPU Pipeline

Nanite

- Hierarchical LOD clusters with seamless boundary
- Don’t need hardware support, but using a hierarchical cluster culling on the precomputed BVH tree by persistent threads (CS) on GPU instead of task shader

Unreal e.g.

Designing

- Entry point
- Application layout
- Window layout
  - Input
  - ->Events
- Renderer
- Render API abstraction
- Debugging support
- Scripting language
- Memory systems
- Entity-component systems (ECS)
- Physics
- File I/O, VFS
- Build System

Game engine is actually dll library
In a solution, add two projects engine(dll) and sandbox(exe)
Sandbox add reference to engine. (link)