**Data Orientation**

POD

No classes, locks, pointers

Entity Component System

Table based, cache efficient

Events through tables, direct copy of GPU,

Data Streaming on tables or queries, and use filters, transforms, generators

Synchronization by thread association – can be stored in the handle, an association list, or calculate through Counter, ObjectIdx, etc

Delay deletion by marking objects as deleted

Handles: {Counter, PoolIdx, union {ObjectIdx16, OBI16, OI32}} or {Counter, Pointer} or Counter

Skip lists, tiled with pointers, tiled compressed

Hierarchical LOD

Components in tables, specialization

Use asserts, do not use exceptions

Search index for component table: handle is an int, hash into a first table, provides index of first hash item in another table. There the items with the same hash are in a linked list.

#typedef HANDLE uint32\_t

std::array<int32\_t> hashed\_index[512]; //use hash(HANDLE) to get index of index list, if -1 then empty

struct component\_t {

HANDLE handle; //handle of the entity

int32\_t prev\_hdl, next\_hdl; //if -1 then no predecessor or successor

component data…

};

std::array<component\_t> component\_table[512];

**Features**

Shadows: ray traced, cube map, Dual Paraboloid Shadow Mapping (DPSM)

Multiple Shadow Maps with Geometric Shader

Billboards, particle effects through geometric shader, fire, rain

Tesselation of terrain and objects

Cube maps

Light probes

glTF 2.0, PBR modular, unified shader

Frustum culling

Earth athmosphere, clouds

Vulkan queries, visibility queries

Post processing

* Lens flare, depth of field, vintage

GUI: Nuklear

Measure events, timing, threads, show in diagrams

Data driven rendering

Updates of trees with dirty bits

Mirror, Stencil Buffer

Water

Transparent Glass

Grass, Trees, Wind, Debris

Occlusion culling

**Renderers**

Simple Forward

* Must run on old Macs, Intel integrated HW

Optimal Forward

* Differential recording, Full parallel recording
* Arrays, shader parameters

Deferred

Ray Traced – RTX

Ray Traced – Compute Shader