# Data Orientation

POD

No classes, locks, pointers

Entity Component System

Table based, cache efficient, acces only via get/set functions, get returns a COPY of the data, syncing via thread ID and HANDLE bits

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.

Simple descriptor binds

for each view {

bind global resourcees // set 0

for each shader {

bind shader pipeline

for each material {

bind material resources // sets 2,3

}

}

}

#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];

Descriptors: Bindless, frequency based, one set per shader stage, slot based (one set per descriptor type like texture, sampler, buffers, …), object type based (one set per object type like light, camera, shadow maps, object maps, …)

# 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, snow

Tesselation of terrain and objects

Cube maps

Light probes

glTF 2.0, PBR modular, unified shader

Bone animation

Videos as textures, stream game graphics as videos, screenshots

Frustum culling, Occlusion culling

Earth athmosphere, clouds

Vulkan queries, visibility queries

Post processing

* Lens flare, depth of field, vintage, HUD, Blood Indicator

GUI: Nuklear

Measure events, timing, threads, show in diagrams

Data driven rendering

Updates of trees with dirty bits

Mirror, Stencil Buffer

Water, normal map blending and animation

Transparent Glass

Grass, Trees, Wind, Debris

Parallax mapping

# Renderers

Simple Forward

* Must run on old Macs, Intel integrated HW
* Frequency based descriptor sets
* Sequential shadow maps

Optimal Forward

* Differential recording or Full parallel recording
* DrawIndexedIndirect
* shader parameters
* Bindless descriptor sets through arrays/buckets, use Push descriptor sets for updating buckets
* Multiple shadow maps with geometry shader
* Ray traced shadows?

Deferred

Ray Traced – RTX

Ray Traced – Compute Shader

References

<http://kylehalladay.com/blog/tutorial/vulkan/2018/01/28/Textue-Arrays-Vulkan.html>

Data-Oriented Design Richard Fabian October 8, 2018 , Fabian, Richard. Data-oriented design: software engineering for limited resources and short schedules (S.1). Richard Fabian. Kindle-Version.

* Tables, existential design

Writing an Efficient Vulkan Renderer Arseny Kapoulkine , in Engel, Wolfgang. GPU Zen 2: Advanced Rendering Techniques (S.227). Black Cat Publishing. Kindle-Version.

Slot based descritpr sets, Frequency based descriptor sets, Bindless descriptor sets, limits on uniform / storage buffers,

Full screen quad without buffers <https://www.saschawillems.de/blog/2016/08/13/vulkan-tutorial-on-rendering-a-fullscreen-quad-without-buffers/>

<http://kylehalladay.com/blog/tutorial/2017/11/27/Vulkan-Material-System.html>