High Level Graphics Programming & VR System Architecture

Hannes Kaufmann

Interactive Media Systems Group (IMS)
Institute of Software Technology and Interactive Systems

Based on material by Dieter Schmalstieg
VR & AR Course Overview

- Introduction

Hardware
- Input Devices
- Output Devices
- 3D Graphics Hardware

Software
- 3D Interaction
- High Level Graphics Programming
- Usability, Evaluations & Psychological Effects
Application Programmer’s View

- Rendering Engine or Scene Graph
- OpenGL or D3D
Low-Level Graphics API

- OpenGL (v 1.0 1992), Direct3D (DirectX 2, 1996)
- Procedural
- Primitives
  - Line, Triangle, ...
  - Color, ...
- Dual Database Problem
  - 1. Representation: Data Objects
  - 2. Representation: Graphical
  - Redundancy, Problem of Inconsistency
High-Level Graphics API

- Rendering Engine (e.g. Unity, Unreal Engine,...) or Scene Graph e.g. OpenSceneGraph, OpenSG, X3D (VRML), Java3D,..
- Object oriented
- Scene Objects – “Objects, not Drawings”
  – Not limited to graphical display
- Interactivity: Event-model for 3D scenes
- Software architecture
  – Toolkit-approach, extendible
Why High-Level API?

- Rapid prototyping
- Rapid application development (RAD)

Interpreted script

Geometry file

Application or Shell (Browser, Editor)

Hi-Level API

Low-Level API
Scene Graph Example: Open Inventor

- Scene graph library
- Based on C++
- Used in research & commercial projects
- Platform & windowing system independent

3. Version: **Coin** by Systems in Motion (SIM), Re-Engineered API, Open Source; ver. 3.0
   http://www.coin3d.org/
Scenegraph – Structure

• Graphical data structure = Scenegraph
• Scenegraph consists of Nodes
• Directed graph! (Head/Tail)
  Directed edges -> Hierarchy
• Use of the hierarchy
  – Semantic Hierarchy: e.g. car (parts)
  – Geometric Hierarchy: e.g. puppet / jointed doll
• Usually one tree is sufficient
• General: Directed Acyclic Graph (DAG)
  – [ Multiple parent nodes allowed ]
  – No directed circles
Scene Graph - Nodes

• Nodes consist of data and methods
• Nodes are of a specific type
  – Type determines behavior
  – Behavior = Reaction to certain events
  – Events are generated by the application – by the user -> Interactivity
• Nodes are instances of a class
  – Scene hierarchy vs. class hierarchy!
• Flexibility: Choose node(type), compose scene graph with nodes
• Extendible: New nodes can be implemented
Scene Graph - Fields

- Attributes (member variables) in nodes are called **fields**
- Fields: set/get, detect changes, connect fields across nodes
- Fields are objects by themselves
  - Float-Object, String-Object etc.

SoMaterial
- ambientColor
- diffuseColor
- specularColor
- emissiveColor
- shininess
- transparency
Example
Graph Traversal: Basic Idea

- Data structure (scene graph) is processed (=“traversed”)
- For each node a number of methods is implemented:
  - Rendering
  - BoundingBox calculation
  - Transformation matrix calculation
  - Handle Events (e.g. picking)
  - Search nodes
  - Write to file
  - Execute user callback...
Graph Traversal Order

- All nodes must be processed
- **In general: Depth-First**
- Traversal uses a State-Engine
- Difference in Group Nodes
  - Ordered Group
    - State Propagation *top-*->down and *left-*->right
    - e.g. Inventor, VRML / X3D
    - Very flexible scene graph generation
  - Unordered Group
    - State Propagation **only top-*->down**
    - e.g. Java3D
    - **Parallel Render Traversal possible** (Threads, SMP)!
State, Stack & Separator

Separation

Red Sphere Separator

Green Separator Cube

Blue Cone

Color state stack

Traversal saves state in Stack

Green

Red
Graph Traversal
Modeling Attributes

- In-between, leaves or fields?

Some toolkits only allow specific structures e.g. X3D Shape & Appearance combined
Transformation-Hierarchy

OpenGL Matrix Stack <--- Transformation hierarchy
Instancing (Re-use)

Example: Car

In case of textures:
• Saves texture memory
Indexed vs. non-indexed polygon lists (e.g. FaceSet):

Non-Indexed:
V={P1=(x1,y1,z1), P2, P3, P2, P3, P4, P3, P4, P5, P6}
F={3, 3, 4}

Indexed:
V={P1,P2,P3,P4,P5,P6}
F={1,2,3,-1, 2,3,4,-1, 3,4,5,6,-1}
Polygonal Shapes: Attribute

Bindings of attributes
• for material, normals, texture coordinates
• specifies mapping of attributes to polygons
• Overall object, per face, per vertex
Dependency Graph

• “Field connections”: Field types must be compatible!

• Two different (overlapping) structures
  – Scene graph
  – Dependency graph (dependent fields)
Engines

- To model complex dependencies in graph
- TargetField := Engine(SourceField)
- E.g. Calculator, Counter, Type converter, Interpolator, Trigger
Node Kits / Prefabs

• Prefabricated sub-scene graphs
  – e.g. transformation, material + shape
  – Simplify the construction of semantically correct scenes
Lab Exercise: Higher Level Programming

• Game-Engine
  – E.g. Unity3D
  – Extended functionality:
    • Tracking input
    • Distribution

– Object oriented programming in C# / Javascript
– Based on an Entity component system (ECS)
Entity Component System (ECS)

- **Entity**: a general purpose object. Usually, only consists of a unique id.
- **Component**: Container object. Raw data for one aspect of the object, and how it interacts with the world. Data is stored in Components.
- **System**: Each System runs continuously (as though each system had its own private thread) and performs global actions on every Entity that possesses a Component of the same aspect as that System.

- Integrated in Unreal Engine, Unity.2019 and others
Entity Component System (ECS)
ECS – Possible Architecture

**Diagram: ECS Architecture**

- **MEMORY MANAGER**
- **LOGGER**
  - **ENTITY MANAGER**
    - IEntity
    - Entity<T>
    - EntityId
  - **COMPONENT MANAGER**
    - IComponent
    - Component<T>
    - Comp. Iterator
  - **SYSTEM MANAGER**
    - ISystem
    - System<T>
  - **EVENT MANAGER**
    - IEvent
    - Event<T>
    - IEventListener
ECS

- Entities are simple compositions, not complex inheritance trees
- Simple object lookups
- Systems could
  - render components
  - detect collisions, compute physics
  - manage health of players
  - Store data in components that they need
- Memory efficient, scalable, extendable
- High performance possible
ECS Further Information

• https://en.wikipedia.org/wiki/Entity_component_system

• https://www.youtube.com/watch?v=ILfUuBLfzGI

Software Design and Components of an VR/AR Framework

Hannes Kaufmann

Interactive Media Systems Group (IMS)
Institute of Software Technology and Interactive Systems
AR/VR Framework: Requirements & Wishes

• Support multiple input & output devices
  – Input: Interface to tracking middleware (e.g. OpenTracker, VRPN)
  – Output: High level graphics programming, Stereo rendering,…

• Handle user interaction

• Allow flexible 3D user interface
  – widget libraries/middleware

• Support of collaboration
  – multiple users, flexible user configuration, mobile work

• Support distributed work

• Easy application design / authoring
VR/AR/MR System Architecture

Application(s)

MR Framework
- Rendering / Event Handling / Multi-user / Distrib.

Tracking Middleware

Distribution Framework

Networking e.g. Session Management

Rendering Engine

GUI

3D UI

Computing Platform
- CPUs, GPUs, ...

Input Devices & Tracking

Output Devices

Network
Example: Distributed VR / AR in Education

- Distributed collaboration over large distances
- Large number of users supported
- Flexible hardware setups
- Interaction depends on input device properties
Distributed Shared Scene Graph

• Shared Memory (SM): Multiple CPUs access the same memory
  – Very simple and popular
  – May need mutual exclusion (locks etc.)

• Distributed Shared Memory (DSM):
  – SM on top of standard message passing

• Distributed Shared Scene Graph: DSM semantics added to a scene graph library
Symmetric Approach: Distributed Shared Scene Graph

Goal: Distribution without programming
Keep existing API intact

- Dual database (app, scene)
- Optimizations

- Distributed shared memory semantics
- Transparent distribution
- E.g.: Avango, Distributed Inventor (DIV)
Updates in DIV

1. App makes update to myField

2. Observer detects change due to notification

3. Network transmission of update

4. Receiver applies update

Synchronisation protocol:
- update field
- create node
- delete node
+ some more for convenience...
DIV - Pipeline

Master

Simulation code

Scene Traversal

Geometry Stage

Rasterization

Display

Slave

Simulation code

Scene Traversal

Geometry Stage

Rasterization

Display

DIV Updates

(sent by Rendering Engine!)
Long Distance Distribution
Requirements for AR Applications

• Main Challenges:
  – Robust application replication
  – Reliable network communication over long distances:
    • Networking Protocols (uni-/multicast) & Bandwidth considerations

• 3 Options:
  – **Input data**: e.g. Tracking data of input devices
  – **Output data**: e.g. Application content, Screen
  – **Intermediate data**: High level metadata for regenerating correct application state
Long Distance Distribution - Example

3 Types of Data:

– Input data: e.g. Tracking data of input devices
  • Tracking Middleware (e.g. OpenTracker, VRPN)
    – For long distance: Use Unicast (UDP) instead of Multicast

– Output data / Application content
  • Distributed Open Inventor (reliable TCP)

– Intermediate data: High level metadata for regenerating correct application state
  • Construct3D: enhanced replication behavior
    – Geometric objects not transmitted! Only high level state data
Example: Distribution - Results

• Platform independent (Windows, Linux)
• Long distance (Vienna - Graz)
• 2-6 machines, 5 app. instances
• Dynamic joining & leaving
• Hybrid networks possible (multicast UDP + TCP)

• Educational evaluation
  6 students (Sir Karl Popper school)
New VR Standards

- OpenXR 1.0 (2019)
OpenXR

- API aimed for developers and a device layer aimed for the VR/AR hardware - abstraction interface with the device

<table>
<thead>
<tr>
<th>Virtual Reality</th>
<th>Augmented Reality</th>
<th>Console VR</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIO</td>
<td>AIO</td>
<td>Console VR</td>
</tr>
<tr>
<td>Oculus Go</td>
<td>HoloLens</td>
<td>PSVR</td>
</tr>
<tr>
<td>Daydream</td>
<td>ML1</td>
<td></td>
</tr>
<tr>
<td>GearVR</td>
<td>ARKit</td>
<td></td>
</tr>
<tr>
<td>Oculus Rift</td>
<td>ARCore</td>
<td></td>
</tr>
<tr>
<td>SteamVR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Mixed Reality</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC</td>
<td>Microsoft</td>
<td></td>
</tr>
<tr>
<td>Facebook</td>
<td>Magic Leap</td>
<td></td>
</tr>
<tr>
<td>Samsung</td>
<td>Apple</td>
<td></td>
</tr>
<tr>
<td>Facebook</td>
<td>Google</td>
<td></td>
</tr>
<tr>
<td>Valve Corporation</td>
<td>ARCore</td>
<td></td>
</tr>
<tr>
<td>Microsoft</td>
<td>Sony</td>
<td></td>
</tr>
<tr>
<td>OS Support</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>Android</td>
<td></td>
</tr>
<tr>
<td>Android</td>
<td>iOS/iPadOS</td>
<td></td>
</tr>
<tr>
<td>Windows</td>
<td>Android</td>
<td></td>
</tr>
<tr>
<td>Linux/Windows</td>
<td>PlayStation 4</td>
<td></td>
</tr>
</tbody>
</table>

- High-performance cross-platform access to hardware
- https://www.khronos.org/openxr/
WebXR API - for Web Browsers

• Allows to develop&host VR/AR experiences on the web
• WebXR provides the following key capabilities:
  – Find compatible VR or AR output devices
  – Render a 3D scene to the device at an appropriate frame rate
  – (Optionally) mirror the output to a 2D display
  – Create vectors representing the movements of input controls

• https://github.com/immersive-web/webxr/blob/master/explainer.md
• https://immersive-web.github.io/webxr/
• https://www.w3.org/TR/webxr/