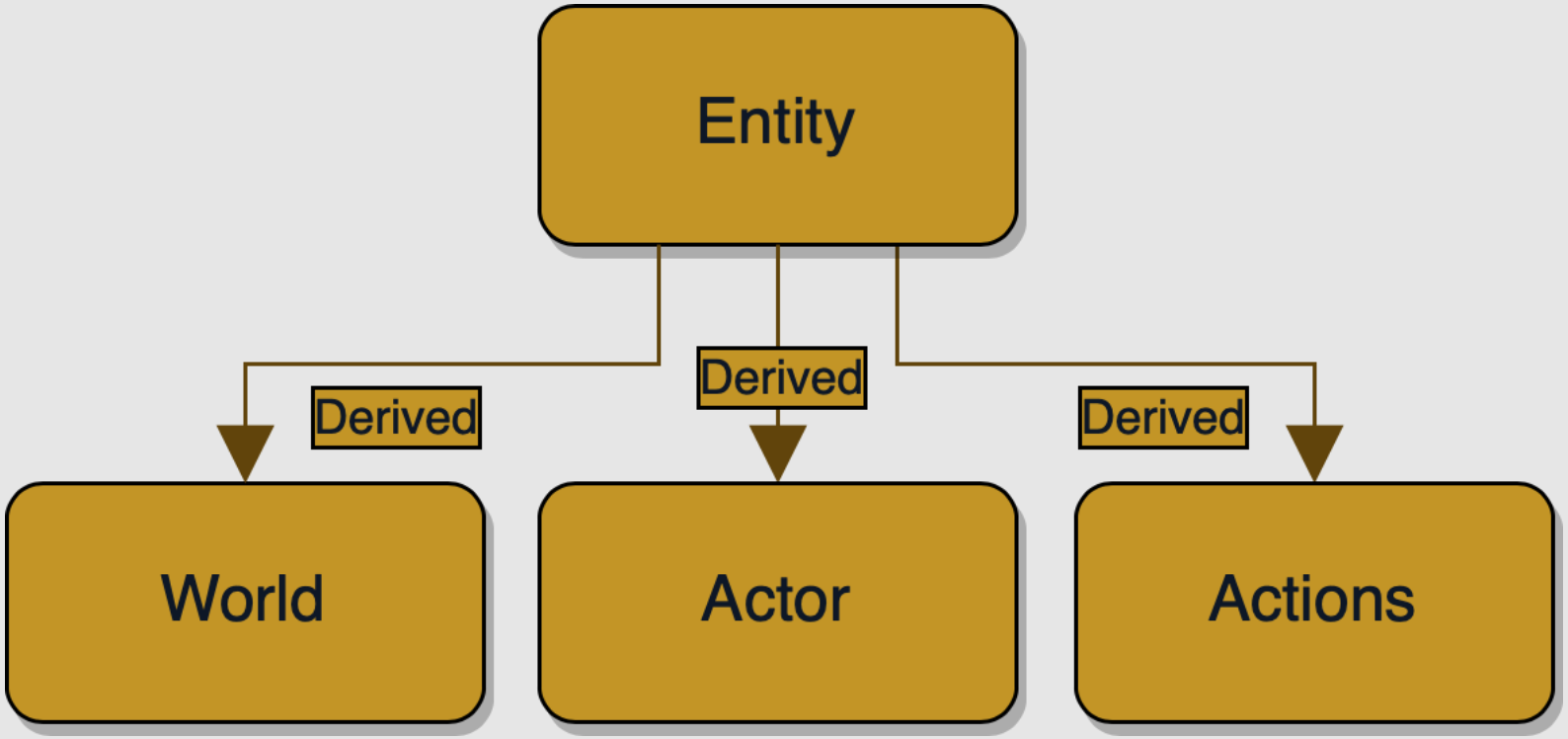
Fiea Game Engine

Logan Harvell

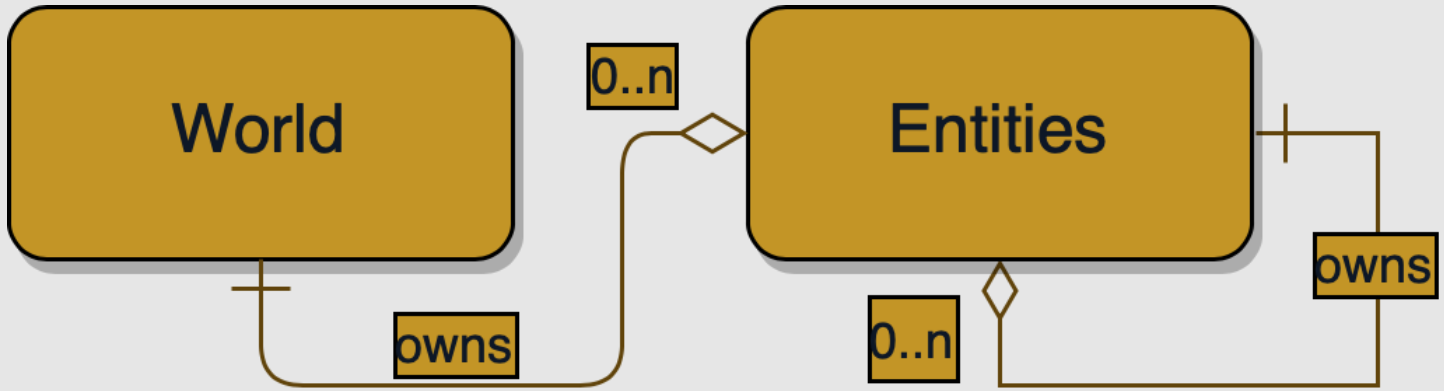
Summary

- Fiea Game Engine is a property-centric game engine
- Supports Windows and Linux projects using Visual Studio
- Data-driven game development using JSON as a configuration language to describe game objects
- Hierarchically structured game objects
- Rendering abstraction layer currently in development

Core Inheritance Hierarchy



Entity Ownership



- The World class acts as the root Entity and manages state and resources
  - Contains the WorldState, used to pass the state down the ownership hierarchy
  - Contains Run, Update, and Stop methods for managing the game loop
- Top level Entity object contained in World can be considered a “level”
  - They contain a hierarchy of objects that will updated during the game loop
  - They can be loaded/unloaded from the World as needed

Transform Class

- Stores these components:
  - Translation, a vector (glm::vec3)
  - Rotation, a quaternion (glm::quat)
  - Scale, a vector (glm::vec3)
- Easy access to each component without requiring matrix decomposition
- Simple transformations only require basic vector arithmetic, preventing unnecessary matrix multiplication
- Matrix compositions are cached to reduce unnecessary matrix multiplications even further

Containers

- Three foundational containers and a container adaptor based on STL
  - SList, a singly linked list
  - Vector, a dynamic array
  - HashMap, a hash table with key-data pairs
    - Keys are hashed for an index into a Vector
    - At each index is an SList chain, storing key-data pairs
  - Stack
    - Container adaptor representing a LIFO stack with Push, Pop, and Top methods
    - The adapted container is specified by a template parameter, defaulted to Vector
- Datum
    - A dynamic array whose type can be defined at run-time
    - Multiple types are supported through a discriminated union
    - Includes int, float, vec4, mat4x4, and references to Scope, RTTI, or other Datum
    - Int, float, vec4, and mat4x4 can are also optionally stored as references

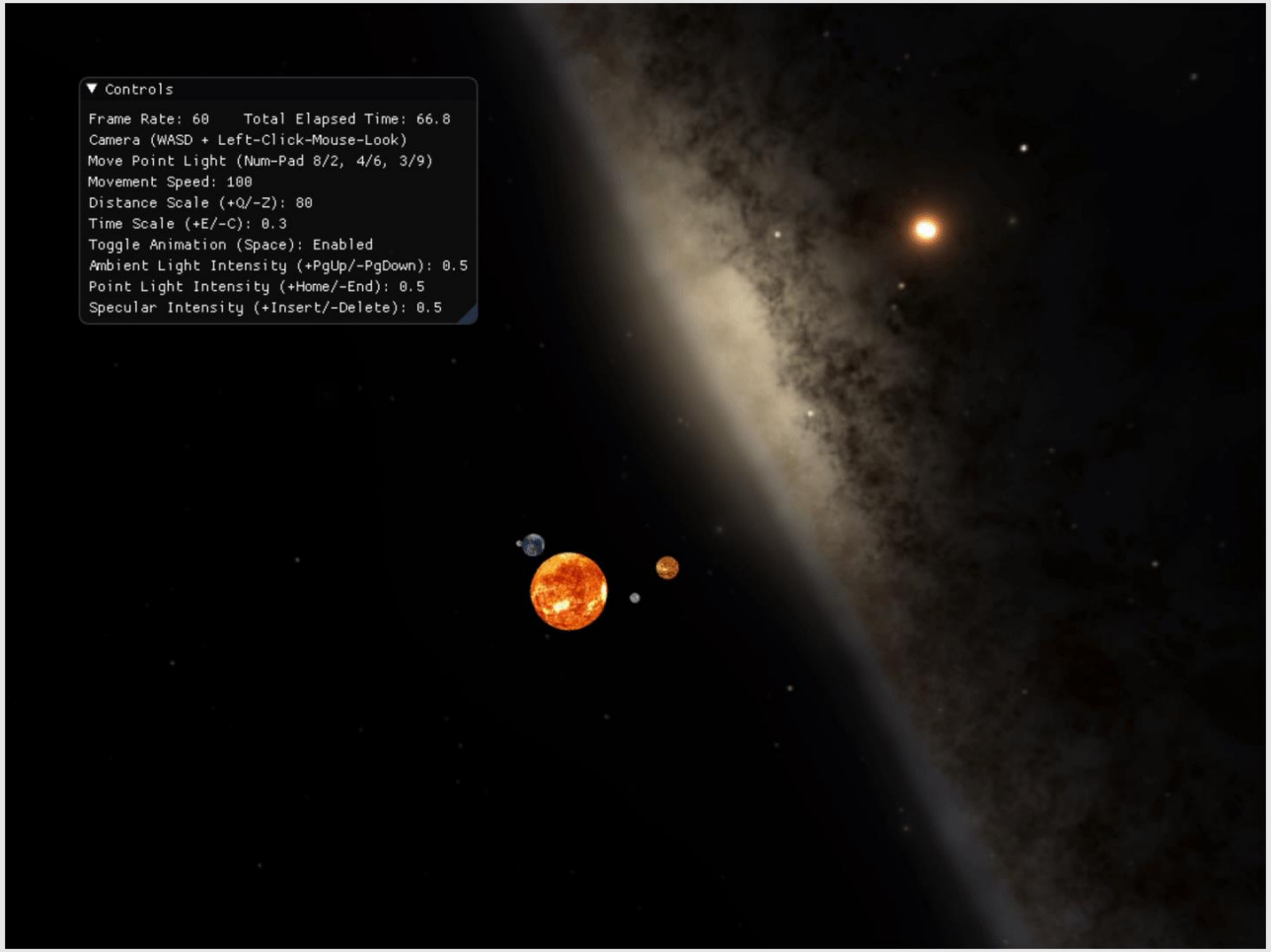
Run-time Type Reflection System

- Scope
  - Represents a table of string keys with runtime definable types, dubbed “Attributes”
  - Implemented as a HashMap of std::string Keys mapped to Datum instances
- RTTI
  - Stores a unique integer ID for each derived class type
  - Enables run-time type information queries
  - Includes TypeId getters, Is/As methods, a virtual constructor, Base type definition, etc.
- TypeManager, registry mapping class types to attribute signatures
- Attributed
  - Base class derived from Scope for registering and exposing attributes
  - Class attribute signatures are registered with the TypeManager
  - Registering signatures allows derived classes to inherit parent attributes
  - Attributes can reference class data members, reflecting the class data structure

Data-Driven Development

- Factory, using the abstract factory pattern
  - Related classes are registered under a base class factory
  - Enables creation of any related class at runtime using a string identifier, a "class name"
- JSON Parser, using the chain-of-responsibility pattern
  - ParseMaster, class that manages parsing a SharedData
  - SharedData, an embedded class that wraps data to be filled with parsed data
  - ParseHelper, a helper class that handles one or more specific parsing cases, i.e. parsing Entity
- JSON configuration files
  - Uses JsonParseMaster with a JsonParseHelper implementation for Entity objects
  - Allows JSON to be used as a configuration file to describe any Entity derived object
  - Uses the factory pattern to create instances of derived classes
  - Can be used to load any Entity from a file, i.e. the World instance, or a "level"
  - Entities can also be re-parsed at runtime to "hot reload" or add attributes

Hot Reloading the Solar System



Hot Reloading the Solar System

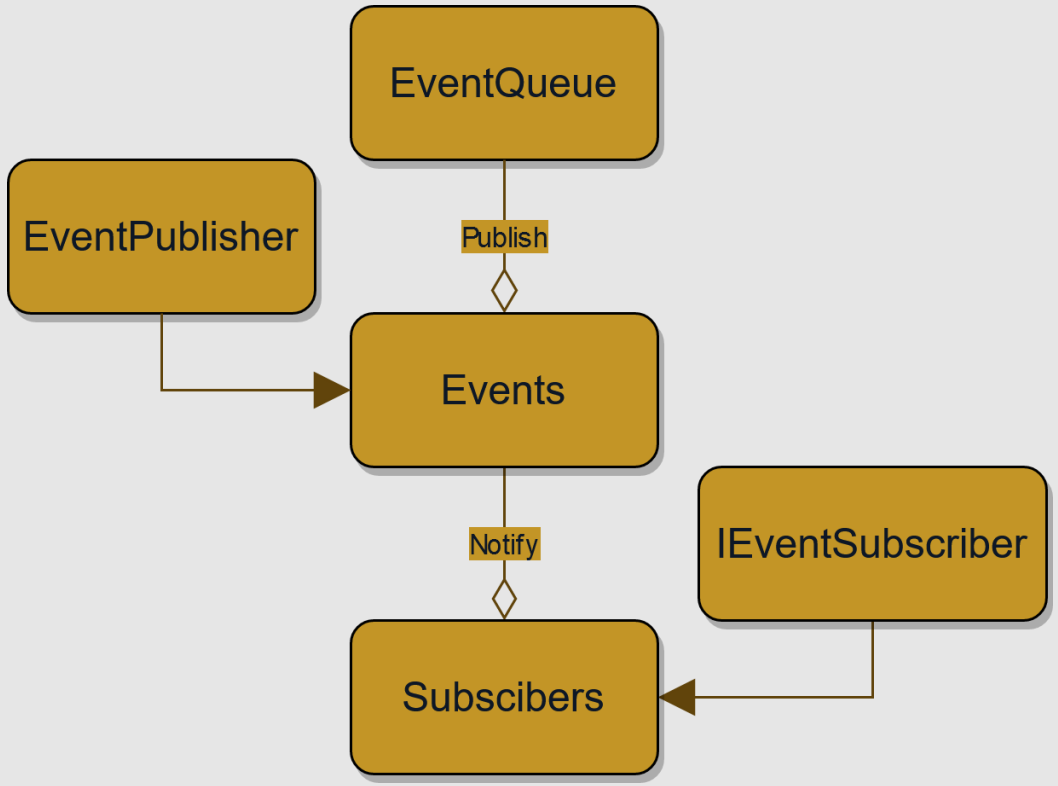
```
{
  "DistanceScale": {
    "type": "float",
    "value": 130.0
  },
  "TimeScale": {
    "type": "float",
    "value": 1
  },

  "Sun": {
    "type": "SolarBody",
    "value": {
      "Diameter": {
        "type": "float",
        "value": 6.0
      },

      "comment": "json for planets below here..."
    }
  }
}
```

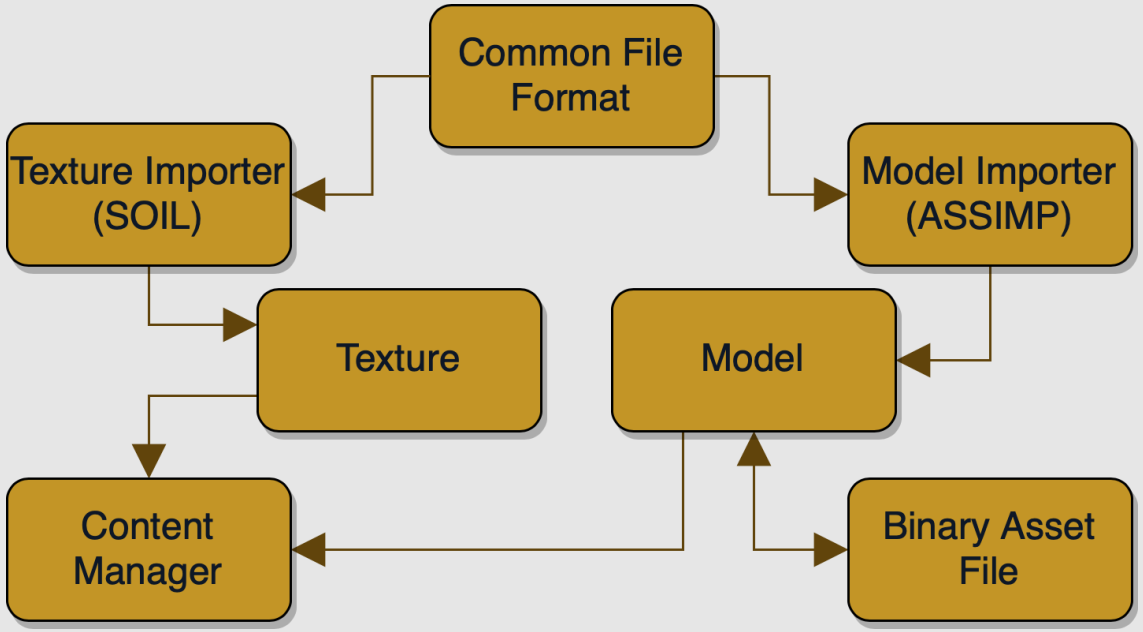
Event System

- IEventSubscriber, interface for event subscribers
  - Subscribers implement Notify to respond to an Event
  - Notify takes a single EventPublisher parameter
- EventPublisher, event Base class
  - Publish method calls Notify on all Event subscribers
  - Passes itself in as parameter for each Notify call
- Event
  - Static methods manage adding/removing subscribers
  - Contains a single data member as a message
  - Message type specified by a template parameter
- EventQueue, manages publishing Events
  - Adds Event instances to a queue with optional delay
  - On Update, iterates through queue and calls Publish on any expired Event
  - EventQueue can also be used to directly Publish an Event



Models and Asset Management

- Model
  - Mesh, the physical geometry
  - ModelMaterial, a texture stack for the model
  - AnimationClip, an animation sequence
  - Bone, a vertex offset within a skeleton of a model used for animat ion
  - SceneNode, a single node in a weighted hierarchy that makes up a skeleton

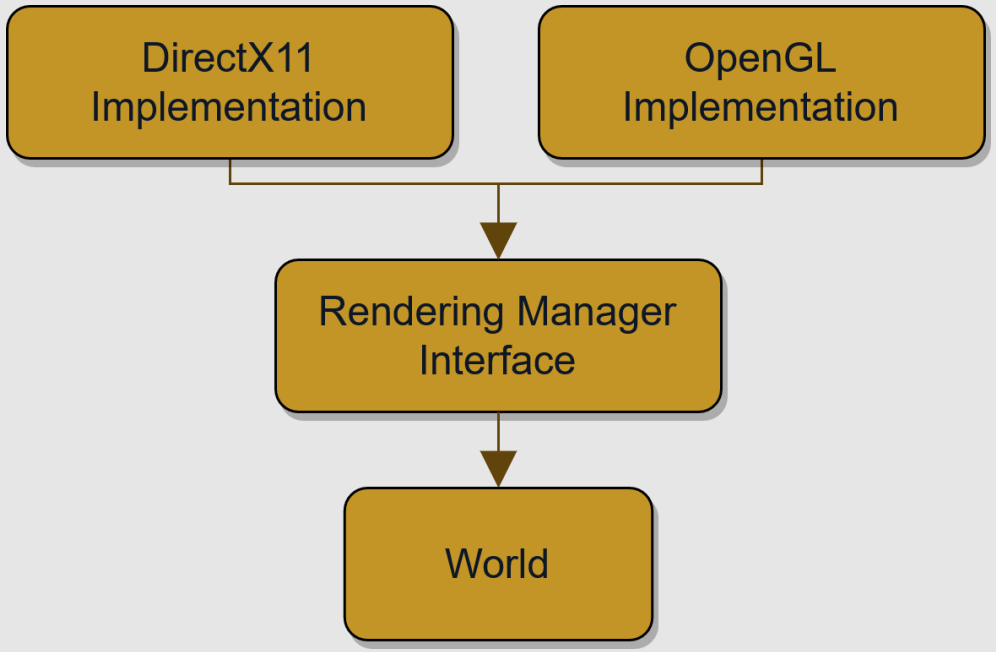


Current Work

- Extending the engine to support the integration of multiple rendering API
- An abstraction layer for rendering is being designed as an interface that can be implemented by any rendering API, i.e. OpenGL, Vulkan, DirectX11/12, and Metal
- This simplifies extending the engine to support specific or multiple different rendering API as needed to best support the end application and target platforms
- The goal is a demo of a dynamic scene rendered using the rendering abstraction layer with an implementation first using OpenGL and DirectX11

Rendering Manager

- Interface that declares structs and pure virtual method prototypes for wrapping rendering API data types and functionality
- Implementations of the interface using a rendering API isolate the dependency
- World maintains a reference to the current RenderingManager instance in the WorldState
- This gives the Entity hierarchy access to the rendering interface without dependencies



Conclusion

- If you would like to follow along with this project, you can…
  - Head to my blog for frequent updates on the progress of the rendering abstraction layer
  - And/or subscribe to the RSS feed
- For more information on the game engine itself, you can…
  - Head to the Fiea Game Engine posting on my website
  - Or explore the source code on GitHub
- Otherwise, feel free to find all the above and more about my musings and tinkerings at https://logantharvell.github.io/