Fabric Engine Overview

Fabric Engine Version 1.2.0-beta
Copyright © 2010-2012 Fabric Engine Inc.

 Copyright 2010-2012 Fabric Engine Inc.

Table of Contents

Chapter 1. Introduction

Fabric Engine is a high-performance processing engine that integrates with dynamic languages by exposing an interface for defining multi-threaded native code operations. Fabric Engine enables high-performance applications to be built using dynamic languages by providing an environment where processor-intensive tasks can be offloaded and processed using muli-threaded native code.

1.1. Ease Of Use

Fabric Engine has been designed for developers who are comfortable building tools and services using dynamic languages. Developers working with Fabric Engine do not require experience of performance software development, nor do they need knowledge of complex languages such as C++.

Developers continue to work with the same tools they are already familiar with, and they can run their applications or services in the usual way. Existing applications and services can be modified to incorporate Fabric Engine for handling compute intensive tasks. Since the majority of code in most applications is not performance critical, this code can continue to be written in the developers' preferred language. No custom IDE or compilation tool chain is required as Fabric Engine integrates a compiler and loads source code from text files.

1.2. Dynamic Environment

Fabric Engine is a completely dynamic development environment where all data structures, graph structures, and code are specified at run time. Fabric Engine can be integrated with any programming language, and is currently integrated with JavaScript and Python. We will add support for other languages over time.

This development paradigm significantly lowers the technical level required to write multi-threaded code, and enables a broad range of developers to build high-performance applications.

1.3. Open Source

The Fabric Engine source code is made available under the AGPL v3.0 license. Commercial licenses are available on request (please email `sales@fabricengine.com`).

1.4. Installation

Developers wishing to build Fabric Engine from source should consult the *Fabric Engine Developer Install Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-DeveloperInstallGuide.pdf] .

Chapter 2. The KL Language

KL is the custom language used to define operators in Fabric Engine. KL combines the benefits of high performance languages like C++ with the benefits of dynamic languages like JavaScript and Python. ‘KL’ stands for ‘Kernel Language’ and refers to the scope of the language: writing operators. Kernels are small stateless blocks of code with an entry function. The syntax of KL is similar to the syntax of JavaScript and C.

2.1. Ease of Use

KL is a language intended to be used by developers of all levels, particularly those comfortable with dynamic languages. Developers familiar with dynamic languages like Python and JavaScript can quickly learn the rules of KL. Developers familiar with static languages will find KL easy to pick up.

2.2. Dynamic Compilation

Fabric Engine integrates the LLVM compiler and uses it to compile the KL source code used in operators. The KL source code is compiled by Fabric Engine as the dynamic language constructs each operator. There are no custom IDE tools required to work with Fabric Engine, so developers can continue to use their normal editors. Applications can use dynamic compilation to create applications that have self-modifying behavior.

2.3. Cross Platform

Fabric applications are completely dynamic and portable across operating systems, CPU architectures and devices. This is because Fabric Engine dynamically compiles KL source code on target, which ensures that there is no compromise on performance. Fabric Engine currently supports all major operating systems on x86 architectures, and support for a wider range of CPU architectures is in development.

2.4. Performance

KL is a strongly typed language like C, and therefore can be optimized in the same way during compilation. The resulting native machine code executes as fast as C/C++ and is integrated into the application dynamically. A benefit of dynamic compilation is that the compilation process generates optimal machine code for the target CPU architecture.

2.5. Security

Fabric Engine can be run in a range of client and server environments. The KL language is constrained in such a way that it makes it impossible for malicious applications to hijack a running application using Fabric Engine. This makes it safe to run Fabric Engine in either client-side (browser-based) JavaScript applications or in online web services.

2.6. Documentation

Complete documentation for the KL language is provided in the *Fabric Engine KL Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-KLProgrammingGuide.pdf] .

Chapter 3. Registered Types

Fabric Engine provides a method for defining types that are used in KL operators and dependency graph nodes. Once a custom type is registered, the data type can be used to populate nodes and can be used in KL operators. This type system enables the types used in the KL source code to be defined before compilation.

Using only a small number of basic atomic types, users can register custom types that combine these types into new complex types. Types can be defined using existing complex types, allowing the definition of complex nested data structures. Types can have KL functions associated with them, enabling object method syntax to be used with the type in KL.

Chapter 4. Dependency Graph

The dependency graph model in Fabric Engine abstracts the concepts of thread management from the user, enabling any developer to create highly scalable multi-threaded applications.

4.1. Multi-Threaded Evaluation

Nodes are constructed using the developer's preferred dynamic language, and the developer defines dependencies between those nodes. Operators are constructed and applied to the nodes in the graph. This combination of nodes, dependencies, and KL operators describes a complete workload made up of many tasks that can be distributed across available compute resources

4.2. Task-Based Parallelism

Each node in the dependency graph can have operators applied to it. Operators define how the data should be processed and when it is propagated through the graph. The binding of an operator to a node represents a task to be executed. During evaluation of the graph the Fabric Engine core can simultaneously evaluate nodes that do not have dependencies on each other.

4.3. Data-Based Parallelism

Each node in the dependency graph can be sliced. Each member contained in a node is duplicated according to the number of node slices. This enables nodes to define homogeneous data sets by storing large quantities of data across many slices. This enables data-based parallelism as operators bound to a node can be invoked for each slice in parallel.

4.4. Dynamic Graph Manipulation

The Fabric Engine core systems can be modified at runtime, meaning that the behavior of a running application can be change based on such things as user input or network events. Between evaluations of the core, the dynamic language can add or remove data, nodes, or operators. The structure of the graph can be changed causing different behavior. A validation is performed after modifications and then execution continues at full speed.

4.5. Event Graph

The Event graph is used to sequence the execution of a set of operators through the construction of a tree structure. While the dependency graph is used for multi-threaded evaluation, the event graph is used for single-threaded evaluation. The event tree is built using a combination of a single event node and a tree of event handlers arranged in a tree structure below it. Evaluation always starts at the event node, then traverses the tree in a depth-first fashion. Operators are evaluated during descent and during ascent.

Event handler nodes in the event tree can have nodes in the dependency graph bound to them. When an event is fired, the nodes in the dependency graph are bound to the event graph and are evaluated along with their dependencies. This system of binding the event tree to the dependency graph defines a dependency between the event tree and any number of nodes in the dependency graph. Firing the event will cause the bound sections of the dependency graph to be updated.

4.5.1. Rendering

The event graph is typically used to build rendering pipelines that draw to the screen using OpenGL. Each viewport in Fabric Engine provides a custom event node that is fired whenever the viewport needs to be rendered. This causes all of the operators in the tree to be evaluated in the sequence that is defined by the structure of the tree.

The structure of the event graph and the sequential evaluation of this graph enable complex rendering configurations to be built. The resulting tree evaluates quickly and can be modified at runtime.

4.5.2. Custom Events

Custom events can be constructed and tree structures built below them. These events can be fired from the host language, which causes the operators in the tree to evaluate and also returns data structures to the host language. This enables tools to query data in the graph that may be distributed across many nodes. In the scene graphs provided with Fabric Engine, the event system is used to compute ray intersections with the geometry in the scene.

4.6. Documentation

Complete documentation for working with the dependency graph is provided in the *Fabric Engine Dependency Graph Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-DependencyGraphProgrammingGuide.pdf] .

 Copyright 2010-2012 Fabric Engine Inc.

Chapter 5. Map-Reduce

Fabric Engine provides a generic map-reduce framework that can be used to create recursively-parallel operations on large data sets. Map-reduce is an ideal paradigm for solving problems that are solved recursively, or require a reduction step such as counting elements in large data sets.

The map-reduce framework can be used in isolation, without construction of a dependency graph. This can simplify the development of applications that do not maintain state, such as web services.

The Fabric Engine implementation of map-reduce expands on the concepts of the classic map-reduce model, providing a more flexible tool. By allowing the results of a map-reduce call do be used to drive further map-reduce calls, problems that are difficult to parallelize become solvable in this multi-threaded context.

Complete documentation for map-reduce in Fabric is provided in the *Fabric Engine Map-Reduce Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-MapReduceProgrammingGuide.pdf] .

Chapter 6. Extensions

Extensions are libraries loaded at initialization time that can define libraries of KL functions and/or bind C++ code libraries and expose their interfaces to KL. The extension system is used to integrate existing libraries of code with Fabric Engine applications. Fabric Engine comes packaged with a wide range of extensions that provide a range of functionality including the Bullet physics library, the OpenCV computer vision library and the Microsoft Kinect SDK.

6.1. Integration of Existing Code Libraries

Fabric Engine provides OpenGL bindings via an extension that uses the GLEW OpenGL wrapper system. Similarly, any API such as Microsoft DirectX or NVidia CUDA can be exposed via custom extensions.

All source code for the provided extensions is available in the Fabric Engine repository and can be used as the basis of a user's custom extensions.

6.2. Documentation

Documentation for the extensions provided with Fabric Engine can be in the *Fabric Engine Extensions Reference* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-ExtensionsReference.pdf] .

Chapter 7. Meta Graphs

The dependency graph, event graph, and operators provide a collection of tools for constructing a wide range of high performance applications. However, sometimes the API needs to be abstracted in order to provide interfaces specific to a given domain, such as image processing or 3D graphics. An abstraction layer can be written in the dynamic language that provides higher level functionality to developers. The abstraction layer can automate the construction of graphs and simplify common tasks.

Fabric Engine provides two such meta-graphs used for 3d graphics. The first is a Python and Qt-based scene graph targeted at industries such as high-end digital content creation for professional 3D industries. The other is a JavaScript scene graph that can be used with our browser plug-in to enable high-performance web applications. Theses meta graphs provide an interface that users familiar with other scene graphs will recognize, which makes it possible to construct scenes in just a few lines of code.

7.1. PyQt (Python) Scene Graph

The PyQt scene graph exposes an interface for building scenes that can be rendered using a custom Qt OpenGL widget. Qt is used for all GUI, layout and window management.

The PyQt scene graph provides systems for Cameras, Materials, Geometry, data loading, and simulation. Using the PyQt scene graph, users can quickly build sophisticated tools and use all of the available Python and Qt technologies available.

The PyQt scene graph is currently in beta. If you would like to take part in the testing program, please email `beta@fabricengine.com`.

7.1.1. Data Loading

A wide variety of data formats are supported by the PyQt scene graph, including Autodesk FBX, Wavefront OBJ, LIDAR, and many image and video formats. The files are opened and parsed by the extension, which returns the parsed data to the calling KL operator code.

7.1.2. Simulation

The PyQt Scene Graph provides a flexible framework for defining custom simulations. The time management of Fabric applications is defined in Python. This enables complex behaviors such as sub-frame evaluation. The Python Scene Graph provides basic acceleration structures and can be customized according to specific use cases.

7.1.3. Rendering

The PyQt scene graph comes with support for forward based rendering, deferred rendering, and volume rendering. An XML based material system uses GLSL shaders to define rendering and can utilize all the features of a modern graphics card. The data driven material system (written entirely in Python) can be modified by users according to their own requirements.

7.2. JavaScript Scene Graph

The JavaScript scene graph is useful for rendering complex data sets, or securely displaying sensitive data online. The JavaScript Fabric Scene Graph supports loading and parsing of binary formats, and provides interfaces to generate custom binary files that encode valuable asset data.

7.2.1. Data Loading

The Fabric NPAPI plugin provides interfaces for opening and writing files on the user's local machine. This enables web applications to write out files from online hosted web applications. Fabric protects the user's security by only exposing this interface via user validated actions such as a file open dialog. For more information, see the *Fabric Engine I/O Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-IOProgrammingGuide.pdf] .

7.2.2. Rendering

The JavaScript scene graph features the same XML based material system as the Python Scene Graph and can also utilize all available features of the users GPU. Through Fabric, users running modern graphics cards can use the latest OpenGL 4.0 features, such as tessellation and geometry instancing.

7.2.3. Documentation

Complete documentation for working with the JavaScript scene graph, along with a collection of examples showing how to set up many common scenes and effects, is provided in the *Fabric Engine JavaScript Scene Graph Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-JSSceneGraphProgrammingGuide.pdf] .

 Copyright 2010-2012 Fabric Engine Inc.

Chapter 8. Getting Started

8.1. Using Fabric in Server Side Applications

Fabric Engine runs in server configurations by integrating with server platforms such as Node.js. The *Fabric Engine Server Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-ServerProgrammingGuide.pdf] provides setup instructions and tutorials for using Fabric for server-side applications.

8.2. Using Fabric in Client Side Web Applications

To build client-side web applications using the JavaScript scene graph, you will need to:

- Follow the instructions in the *Fabric Engine JavaScript Developer Install Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-JSDeveloperInstallGuide.pdf] to get set up for development using the NPAPI browser plugin.
- Once you have the JSSceneGraph source code, you can browse the samples in the /Samples folder, and run them using your own local web server. Many of the demos in the 'BasicDemos' folder are provide simple examples of how to set up basic scenes.
- Read the documentation provided on how to work with the JavaScript scene graph interfaces in the *Fabric Engine JavaScript Scene Graph Programming Guide* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-JSSceneGraphProgrammingGuide.pdf] .
- Use the browser's debugging features. Browsers such as Chrome and Firefox come with powerful debugging tools that enable developers to inspect running applications and understand the code execution. See *Google Chrome Developer Tools* [http://code.google.com/chrome/devtools/docs/overview.html] and *Firebug* [http://getfirebug.com/] (for Firefox).
- Fork the Fabric repository and build applications in your own repository. You can then propagate bug fixes and updates from the Fabric Engine repository into your own. Follow the instructions on GitHub to fork your own repo: `http://help.github.com/fork-a-repo/`

8.3. Using Fabric in Python Qt Applications

The Fabric Engine Python Scene Graph is still in closed beta.

To gain access to the Python Scene Graph, please request access by emailing `beta@fabricengine.com`.

8.4. Building Fabric from Source

Fabric Engine can be built from source. The following guide provides instructions on setting up your build environment and running the build scripts: *Building Fabric Engine from Source* [http://documentation.fabric-engine.com/1.2.0-beta/FabricEngine-BuildingFabricEngineFromSource.pdf]