Dojo GFX 2.0

Unified 3D Graphic Web

Dojo Toolkit

Version

Number	Date	Notes
1.0	TBA	Initial version.

Table of Content

Table of Figure

Introduction

About more than three years since Apple firstly announced canvas HTML element, web inventors have been working hard to evolve HTML visual aspects, start from two dimensions (2D) to three dimensions (3D) graphic visual. The problem is they create new inventions without forming any standard rules. It happened when Apple used their new HTML element (canvas) instead of supporting Scalable Vector Graphic (SVG) standard. Some complained for their decision and until now the diversity between HTML layout engines become worse and worse. DOJO Toolkit saw the canvas implementation diversity and implemented GFX 1.1 as their respond to the problem. Now, GFX 2.0 will do the same like its ancestor (GFX 1.1) to solve canvas implementation diversity in 3D. This document concentrates on three parts. One is scope of GFX 2.0 application usage in real situation. Another is explaining goals of the GFX 2.0. The other explains about abstract design of GFX 1.1 and GFX 2.0.

Scope

In term of project scope, new GFX (or GFX 2.0) library covers both 2D and 3D graphic environment. It focuses on 3D development and provides a way to support 2D (since 2D is subset of 3D graphics and the plan is to build on top library to deal with 2D, which will be explained in detail). GFX 2.0 will focus on animation, by all means modifying transformations (translation, rotation and scaling) for specific object.

Goals

The GFX 2.0 has couple goals:

- Unified
  GFX 2.0 library shall provide unified JavaScript (JS) Application Programming Interface (API) that uses any of Canvas3D (or 3D Canvas) implementations. At the moment, there are couple different implementation of Canvas3D, which are Firefox Canvas3D 1.1, Firefox Canvas3D 2.0 (addition to version 1.1 which supports shader features) and Opera Canvas3D. They have different API to be used and it would be painful to developer to provide different set for each version of Canvas3D. The great benefit of this feature to developer is able to display what they developed using GFX 2.0 in different 3D drawing engine of Canvas3D with minimum changes in code and graphics.

- User friendly
  Another feature from this library is providing a useful set of helpers for developer. The current API of Canvas3D libraries (both Firefox and Opera) are too hard to be used (too many low level graphic functions). Hence, GFX 2.0 shall provide a set of handy functions includes:
  - Easy environment setup.
  - Rich shape builder (such as box, cones and loading 3D model files).
  - Flexible view manipulation.
  - Easy interaction between GFX 2.0 and outside environment.

- High performance
  GFX 2.0 shall consider performance as well since it needs to be run in high speed to reduce glitch effect. Some point that can be considered includes:
  - Optimized way to calculate transformations (camera or object transformation).

- Optimized way to know when GFX 2.0 needs to render the environment (for example, it does not render when no state changes includes web page scrolling, mouse hover, windows move or camera movement).
- How to make use of server memory (if it is available) called GPU (Graphic Processing Unit) memory or VRAM (Video Random Access Memory).

Real World Practices

In real example world, it covers a wide range of graphic applications includes:

- Model Viewer
  GFX 2.0 supports user to load or create 3D model in 3D environments. For example, a furniture website would like to show 3D model furniture to user. User is able to interact with the 3D model furniture like manipulating the view (rotate, zoom or pan view) and changing 3D model state (opening door of wardrobe).
- Statistic Chart
  GFX 2.0 allows user to view charts both in 2D and 3D environments. In 3D environment, user can manipulate view to be rotated and zoomed.
- Mathematic Graph
  GFX 2.0 also can be used to display mathematic equation graph in 2D and 3D. User can analyze the equation shape or specific point location.

GFX 1.1 and GFX 3D

GFX is part of DojoX that an area of development and extensions to the Dojo toolkit. Currently, GFX 1.1, which released by Eugene Lazutkin and Kun Xi, only supports 2D graphic environment. At the moment, support for 3D environment is backed up with a library called GFX 3D, which implemented by Kun Xi.

In details

GFX 1.1

There is similar purpose between GFX 1.1 and GFX 2.0, which make unified JS interface among different APIs. The different is GFX 1.1 for 2D graphic API (SVG, VML, Silverlight and Canvas) whereas GFX 2.0 for 3D graphic API.

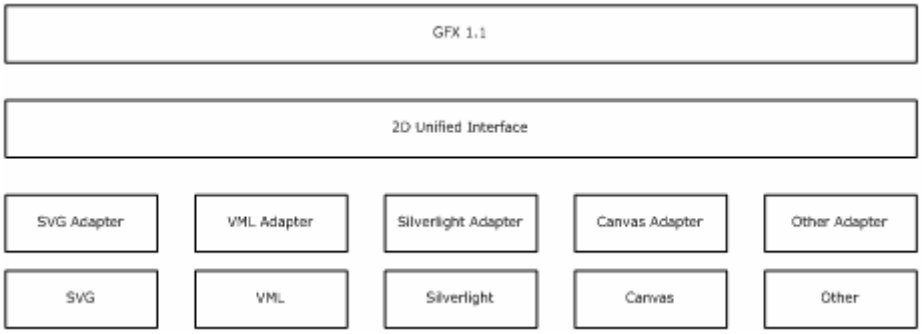

Figure 1: GFX 1.1 Bird-View Implementation

As figure 1 describe, the 2D unified interface eliminates coupling between GFX. In the real JS implementation, the developers tried to optimize the library by not doing prototyping for unified interface, what they did is direct implementing the supported functions into each drawing engine implementation, for the non-supported functions,

JavaScript will automatically return null value when user tries to call them. When GFX 1.1 is initialized, user requires selecting one of drawing engine so that GFX 1.1 will assign appropriate function to the system.

GFX 3D

Interestingly, GFX 3D has no native 3D render engine (like Firefox Canvas3D relies on OpenGL engine), so it only projects 3D object into 2D shapes by relying to GFX 1.1 library. Besides mimicking 3D shape, Kun Xi also tried to hack around the lightning, coloring, and texture to visualize 3D environment.

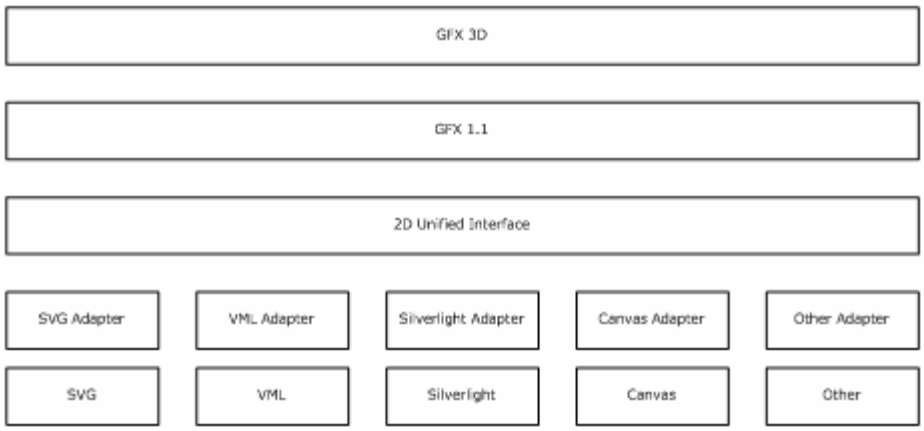

Figure 2: GFX 3D Bird-View Implementation

Advantages

- This library uses widely known drawing engine, which in majority, users are able to use it.
- No need native 3D render engine to display simple 3D graphic environments.

Disadvantages

- Need huge effort to emulate 3D using 2D library where there are cross platform libraries have done that (OpenGL).

GFX 2.0

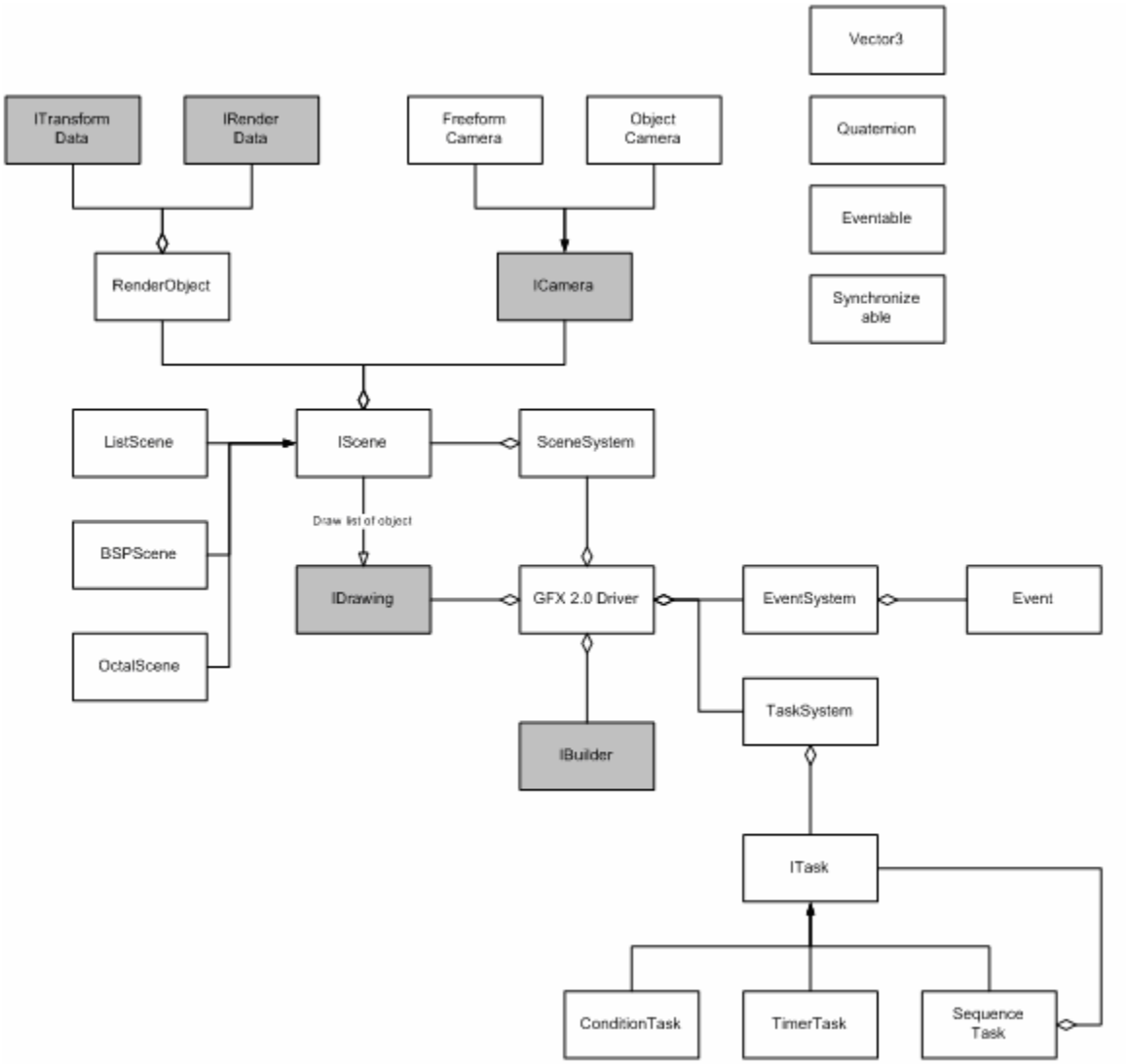

Figure 3: Bird-View GFX2.0 Design

From Figure 3, there are shaded class (IDrawing, ICamera, IRenderData, ITransformData and IBuilder) mean they need to be derived for each Canvas3D implementation. For additional information, this design uses Object-Oriented features (inheritance and interface) in order to make readers to understand easily. The problem is JavaScript does not support them so in the next document it will be explained the implementation of this library and reader can use this document to compare.

IDrawing

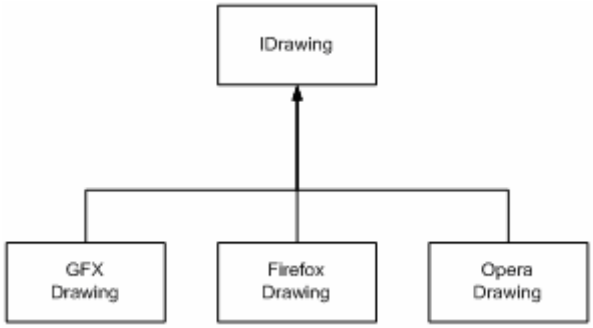

Figure 4: IDrawing Design

This interface is responsible to provide unify API for GFX 2.0 render system. It does not matter whether drawing engine has 3D render system or not as long as it can provide basic functions to:

- Draw triangle shape.
- Draw texture as image.

It is obvious that Firefox Canvas3D and Opera Canvas3D satisfy all the requirements above; the problem is located in the GFX 1.1 library whether it can produce "similar" product.

Some predicted functions for IDrawing components:

- Initialize
  This function is to setup specific drawing implementation for graphic environment.

- Draw
  This function represents one render cycle for each specific implementation. An example of normal render cycle is:
  1. Clear previously drawn frame or scene.
  2. Set background color for the environment.
  3. For each RenderObject in the list:
     a. Set object transformation (translation, rotation and scaling) from current object value.
     b. Draw triangle shapes according to vertex data.
     c. Set normal according to normal data.
     d. Set color according to color data.
     e. Set texture according to material data.
     f. Recursively doing step 2 for each RenderObject children in the current object.
  4. Clear any temporary variable used.

  Note that, for each implementation, it may have completely different cycles. Let's take example for drawing engine which it uses 3D native render engine where it uses buffer in the server memory (GPU VRAM).

IRenderData

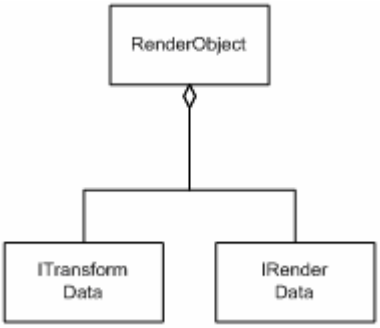

Figure 5: IRenderData Relation

This interface is responsible for make abstraction for data rendering. Previously, IDrawing is for rendering method, IRenderData is special for container to hold data for rendering. For each specific implementation, this interface may include:

- Vertex Data
  This data describes vertex position.

- Color Data
  This data describes color for each vertex data element.

- Normal Data
  This data describes normal vector for each vertex data element.

- Material Data
  This data describes material for the object.

- Shader Data
  This data describes program for vertex and fragment shader of object.

RenderObject

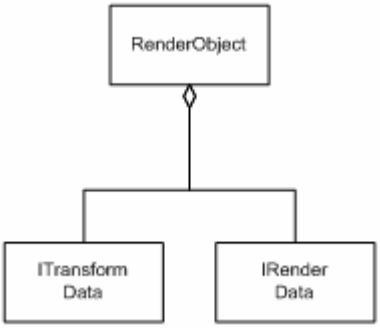

Figure 6: RenderObject Design

This class represent object that can be drawn into 3D environment. It should have a way to determine object transformation (it is called ITransformData, which will be explained in detail) and some other information such as name to identify which object is which. This class also can have children to introduce hierarchy object hence it is necessary to have relative transformation rather than absolute transformation.

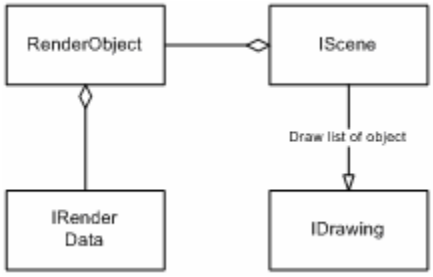

Figure 7: Partial Rendering Connection

The reason why rendering data is put separated from RenderObject because one IRenderData object can be referenced by two or more RenderObject to eliminate duplication data. Note that, when IRenderData data value is modified then all instances RenderObject that refer to that particular IRenderData will be changed as well.

ITransformData

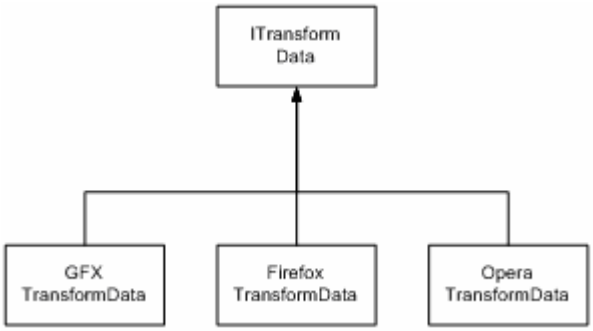

Figure 8: ITransformData Design

ITransformData is special abstract class to deal with object transformation. It has to cover four different transformations; they are translation (uses vector three dimensions, called Vector3), rotation (uses quaternion system), scaling (uses vector three dimensions) and specific transformation data structure for drawing engine. Since each 3D render engine uses different transformation method, it is good idea to provide its format to have better performance. In the ITransformData, it should provide an indicator variable whether transformation is changed and functions to update transformations into specific drawing engine transformation. When object is being processed to be rendered, it will check whether the transformation is changed or not; if yes, then call functions to update the specific drawing transformation and use it; if no, then render engine just straight away use the calculated drawing transformation.

IBuilder

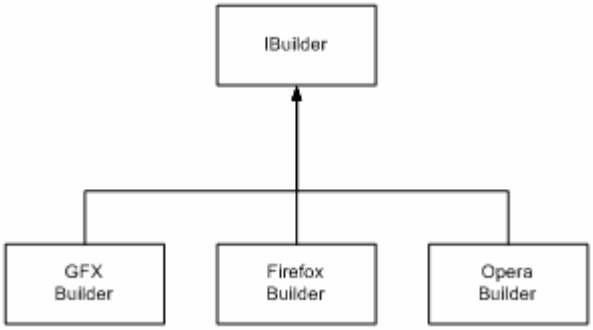

Figure 9: IBuilder Design

This interface has role to create object for couple classes:

- ITransformData
- IRenderData
- RenderObject

It is high recommended to use this factory instead of using constructor for each class that described. This practice is favorable to reduce coupling between GFX library and user code.

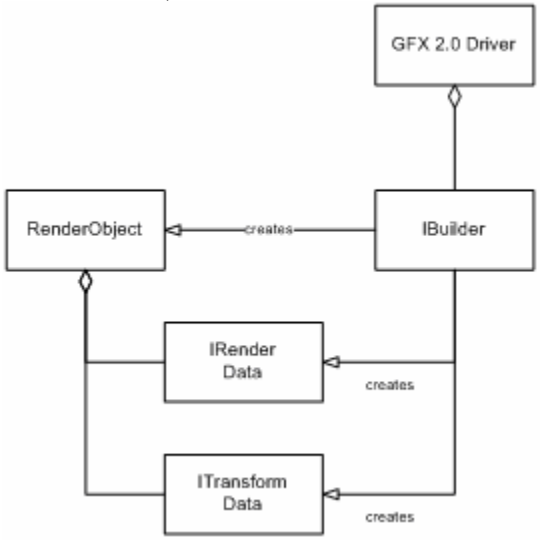

Figure 10: Object Creation Relation

Sequence of function to be performed when user wants to create RenderObject in environment:

1. User gets IBuilder from GFX 2.0 Driver.
2. User creates ITransformData by calling method CreateTransform from IBuilder.
3. User modifies ITransformData to set transformation for object (set translation, rotation and scaling).
4. User creates IRenderData either:

a) Using pre-defined shape definition by calling CreateShape and specifies what shape you want.
b) Using empty definition by calling CreateRenderData and specifies vertex, normal, color and other data.

5. User calls CreateRenderObject from IBuilder and put IRenderData and ITransformData as argument.

Note, that process only created the RenderObject. It has not been attached into IScene (RenderObject container of the 3D environment).

ICamera

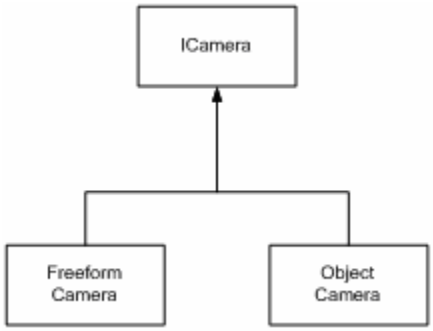

Figure 11: ICamera Design

This interface manages to control the camera view. It has two different type of camera:

- Freeform Camera
  This is standard camera utility. It has panning, rotating, zooming functions.

- Object Camera
  This is more advanced camera for dealing with object class. It has following object function, orbiting object function.

ITask

ITask is an innovative component to help modify 3D environment. User can provide a task (series of actions) and let TaskSystem (manager to handle the tasks) manages the task as criteria are described.

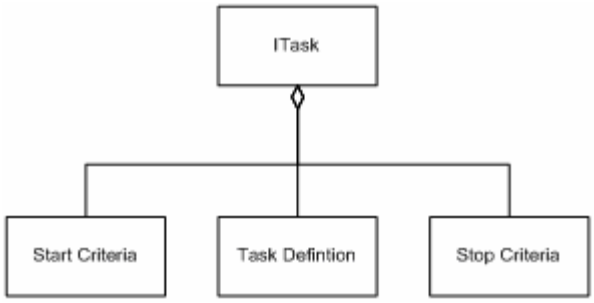

Figure 12: ITask Definition

ITask consist of three components, start criteria (the requirement to run the task) and task definition (series of actions) and stop criteria (the requirement to stop the task and proceed to next task).

Name	Type	Arguments	Return
Start Criteria	Function	None	Boolean
Task Definition	Function	None	None
Stop Criteria	Function	None	Boolean

Figure 13: ITask Elements

As described in above, all ITask components are actually functions with described argument and return types.

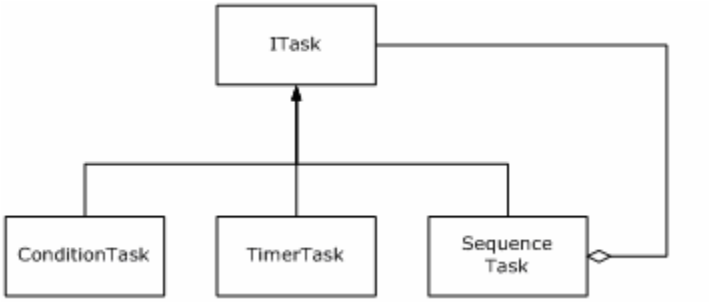

Figure 14: ITask Design

There are couple different types of task:

- Condition Task
  This task will be executed as long as condition that supplied is satisfied. For example, creating task for specific object when out of specified range, it will be removed.

- Timer Task
  This task will be executed as long as time that specified already expired. For this task, it needs special care when user changes scene object because if the time calculation only subtract current time with task created time then it may behave incorrectly. The simple solution is to put a special time inside TaskList to say time when IScene is switched and the time calculation become subject current time with switching time of IScene. This task should provide option to decide whether do this task in limited or unlimited time.

- Sequence Task
  This type of task has multiple sub-tasks that need to be executed. It checks current item in the sub-tasks list, if the sub-task requirement is satisfied then execute the sub-task and decide to remove the current sub-task or put it into last in the sub-task list.

TaskSystem

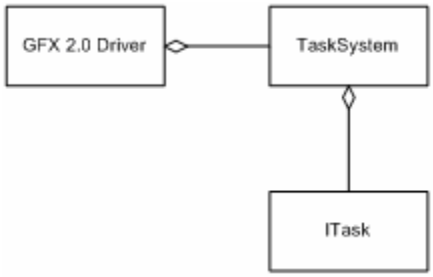

Figure 15: TaskSystem Design

TaskSystem is responsible to manage series of tasks in the current active TaskList object. In TaskSystem execution, it will iterate every task from beginning to end of the list. Firstly, TaskSystem checks the StartCriteria (from ITask), if satisfied the criteria then task will be moved into special place in order to be executed and some requirements such as interval execution, stop criteria and task definition, will be taken account by TaskSystem. TaskSystem will run the

current task (basically run the task definition) in specific intervals until the StopCriteria is satisfied. Then, TaskSystem will discard the current task and check for the next task.

Examples of Scenario:

- Scenario one:

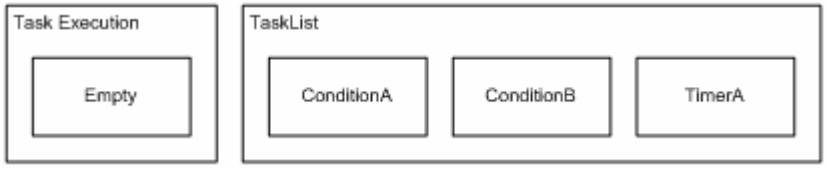

Figure 16: Scenario-01

Sequence of TaskSystem execution:

1. TaskSystem checks ConditionA task (imagine it does not satisfy the start criteria).
2. TaskSystem checks ConditionB task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).
3. TaskSystem checks TimerA task (imagine it does not satisfy the start criteria).

Figure 17: Scenario-01 Update-01

4. TaskSystem checks ConditionA task since TaskSystem was checking the last item on the list (imagine it does not satisfy the start criteria).
5. TaskSystem checks TimerA task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).

Figure 18: Scenario-01 Update-02

6. TimerA's StopCriteria is satisfied then it is removed from Task Execution.
7. TaskSystem checks ConditionA task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).

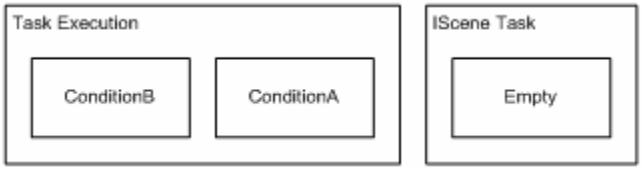

Figure 19: Scenario-01 Update-03

8. ConditionB's StopCriteria is satisfied then it is removed from Task Execution.

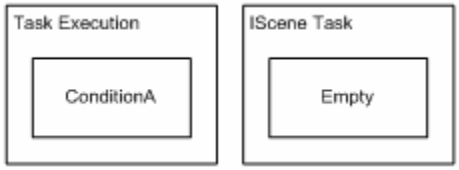

Figure 20: Scenario-01 Update-04

9. ConditionA's StopCriteria is satisfied then it is removed from Task Execution.

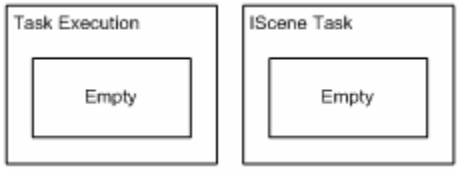

Figure 21: Scenario-01 Update-05

- Scenario two:

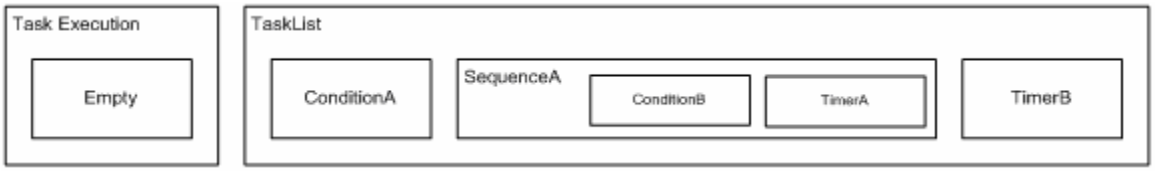

Figure 22: Scenario-02

Sequence of TaskSystem execution:

1. TaskSystem checks ConditionA task (imagine it does not satisfy the start criteria).
2. TaskSystem checks SequenceA with task ConditionB as current task (imagine it does not satisfy the start criteria).
3. TaskSystem checks TimerB task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).

Figure 23: Scenario-02 Update-01

4. TaskSystem checks ConditionA task since TaskSystem was checking the last item on the list (imagine it does not satisfy the start criteria).
5. TaskSystem checks SequenceA with task ConditionB as current task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).

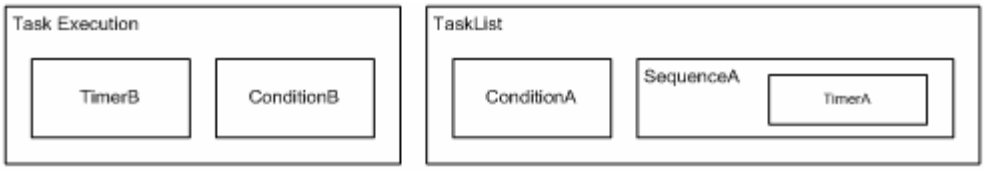

Figure 24: Scenario-02 Update-02

6. TimerB's StopCriteria is satisfied then it is removed from Task Execution.
7. TaskSystem checks ConditionA task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).
8. TaskSystem checks SequenceA with task TimerA as current task (imagine it does not satisfy the start criteria).

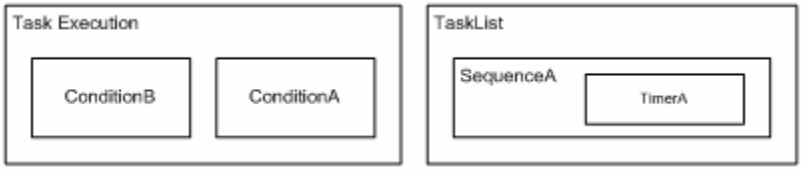

Figure 25: Scenario-02 Update-03

9. TimerB's StopCriteria is satisfied then it is removed from Task Execution.
10. TaskSystem checks SequenceA with task TimerA as current task (imagine it does satisfy the start criteria, then it is moved to task execution and it is executed in specific interval time until stop criteria satisfied).

Figure 26: Scenario-02 Update-04

10. TimerA's StopCriteria is satisfied then it is removed from Task Execution.

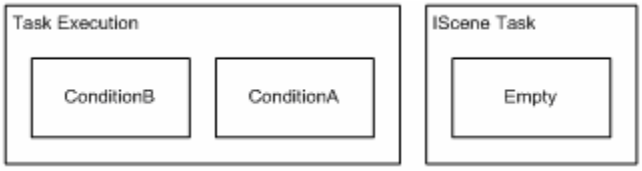

Figure 27: Scenario-02 Update-05

11. ConditionB's StopCriteria is satisfied then it is removed from Task Execution.

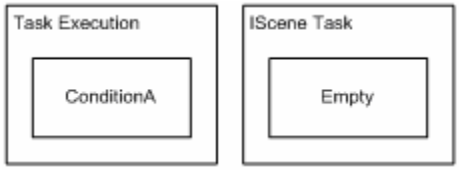

Figure 28: Scenario-02 Update-06

12. ConditionA's StopCriteria is satisfied then it is removed from Task Execution.

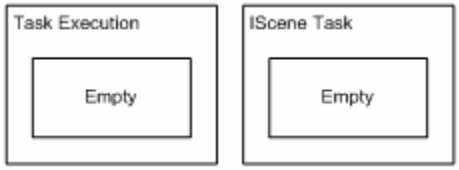

Figure 29: Scenario-02 Update-07

EventSystem

This component deals with event management in GFX 2.0. It connects JS event handling with GFX library. For example, when JS mouse over event is raised then EventSystem will execute all event handler that defined by user.

Figure 30: EventSystem design

In EventSystem, it accepts three different event types:

- Mouse events (onMouseUp, onMouseDown, onMouseMove, onMouseClick, onMouseIn and onMouseOut).
- Keyboard events (onKeyDown, onKeyUp and onKeyPress).
- Custom events, (other events that not in previous categories, user defined events).

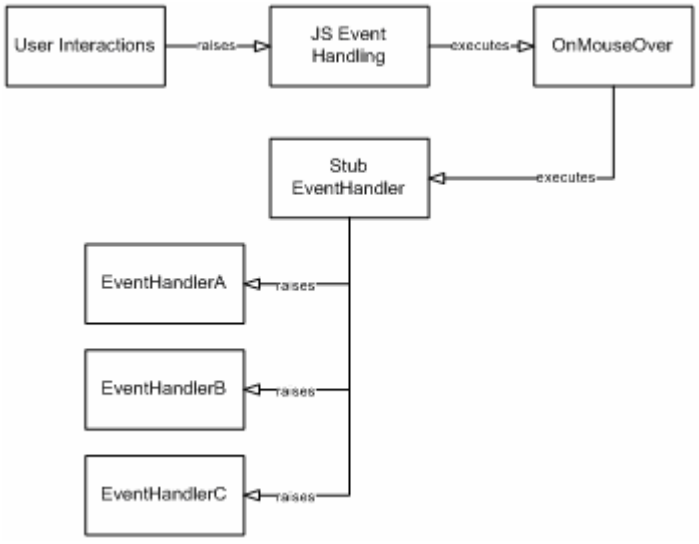

Figure 31 Event Execution Flow

The EventSystem actually create a stub event handler that executes all functions that included. As above figure describe, once user does interaction (moving his mouse), JS will catch the event and run the stub event handler (function that defined fro OnMouseOver in JS) and run all event handler for that event (in this example, they are EventHandlerA, B and C). It is same case, when user wants raise custom event by calling function RaiseEvent in EventSystem with specifying name of the event, it will run the stub event handler and run all event function inside EventList.

IScene

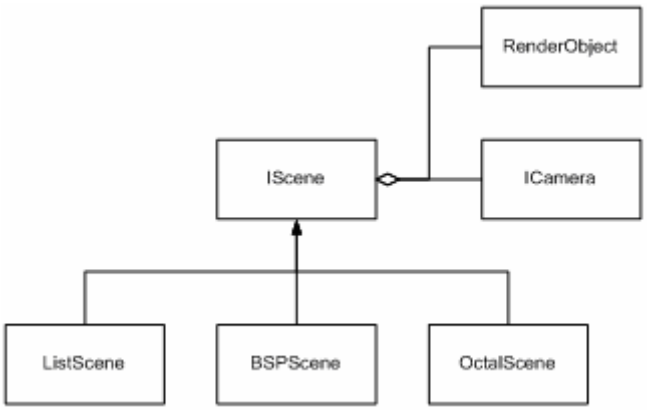

Figure 32: IScene design

IScene is an interface that represents to container in the graphic environment. It has collection of RenderObject to be drawn and ICamera to control the view of the environment. It has couple different implementation for this interface to specify the way of rendering the environment.

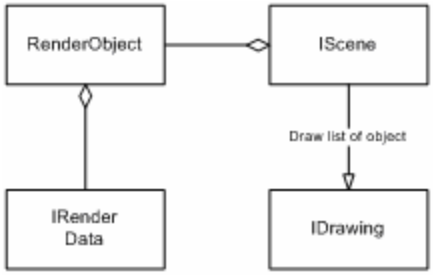

Figure 33: IScene Rendering

For each IScene implementation should define Draw function to manage objects inside the IScene into a queue let IDrawing to process the queue.

ListScene implementation uses array data structure to hold all the RenderObject. It does not need to do anything special when managing objects into queue. This means object that not suppose to be drawn (due to object is out of range from current view boundary) is included into the queue hence it decrease the performance.

BSPScene and OctalScene could be implemented for future development to increase the performance of rendering.

GFX 2.0 Driver

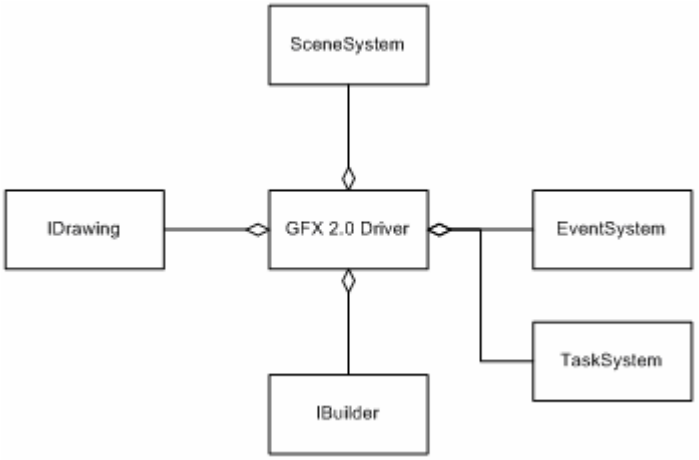

Figure 34: GFX 2.0 Driver Design

This class is the main point of the GFX 2.0 library. It is the place where user can decide which back-end technology and make use of system that available in the library. It is highly recommended to use GFX 2.0 library from this point to decrease tight coupling between library implementation and user code.

Conclusion

In this document, all explanations use Object-Oriented feature to explain design about GFX 2.0 infrastructure. The design will be backed-up with next document that explains more specific for JavaScript environment (removing interface class aspect, optimization functions and methods for each class).

References

Dojo toolkit website, http://dojotoolkit.org/
DojoX.GFX 1.1 website, http://dojotoolkit.org/book/dojo-book-0-9/part-5-dojox/dojox-gfx
DojoX.GFX 1.1 documentation, http://docs.google.com/View?docid=d764479_9hgdng4g8
DojoX.GFX3D documentation, http://www.kunxi.org/files/before-we-code-we-document/intro-dojox_gfx3d/index.html
Eugene Lazutkin, http://lazutkin.com/blog/
Kun Xi, http://www.kunxi.org/