Introduction

**Note:**

## Three Research Fields of Computer Graphics

As depicted by following figure, any software related to computer graphics can be divided into three parts: Design, computation and rendering.



**Design:** Geometry objects definition by equations.

Some geometry objects, such as sphere, cylinder, cone, torus, have equations. For example, we can define a sphere by equation x2+y2+z2=r2.

Most geometry objects have no equations. For example, human faces, mountains, trees, grasses, animals, smoke, cloud, .etc.

If we cannot find an equation for descripting an object, we must create some equations for that. For example, we can define a B-spline curves by 16 vertices, and employ many tiny B-spline patches that surrounding an object for its geometry definition. Often we employ triangles instead of B-spline curves for object surrounding primitive.

**Computation:** computing physics values on surface or inside of geometry object through numerical method.

Physics are also descripted by equations. In most case, they are partial differential equations. For example, Maxwell equations for descripting electromagnetic field and Navier-Stokes equation for descripting fluid.

Very seldom we can get analytic solution for these partial differential equations. We can only compute physics values through numerical method and get their numerical solution. To do this, design software in figure above divides geometry objects into tiny voxels and change partial differential equations into approximate linear equations for these tiny voxels. By solving the approximate linear equations, we can compute approximate numerical physics values on and inside geometry object.

**Rendering:** Create image of geometry objects in a scene and output the created image on screen.

Rendering is performed by GPU on graphics adapter. GPU can only processing graphics primitives, such as point, line section, triangle, etc., CPU-side applications create graphics primitives from voxels, and transfer the primitives to GPU. GPU-side shaders and fixed-function pipeline renders the primitives, create image from them, and output the created image on screens.

No matter what type of primitives are rendered, GPU defines primitives by vertices. A point is defined by one vertex, a line section is defined by two vertices, and a triangle is defined by three vertices. A vertex has several attributes, including position, normal, ID, material, texture coordinate, .etc.

Shaders are programs that run on GPUs for rendering, different geometry objects may employ different rendering shaders. WebGL 1.0 employs OpenGL ES 2.0 shader language for rendering.

**The main function of a WebGL engine is to renders graphics primitives, not define geometry objects or does computation.**

## Web Architecture

Web architecture can be illustrated as following figure.



Web has two types of computers.

One is server-side computers that run Web Application Servers, such as Tomcat, WebSphere, WebLogic, Internet Information Services (IIS), .etc. A web Application server processes client’s http requests, and responses processing results to the clients. The other is client-side computers that run browsers, such as Chrome, Firefox, opera, safari, Edge, IE, .etc.

Both sides execute programs and implement specific functions. WebGL engine must decide which functions are located on servers, which functions are located in browsers, as well as how they cooperate each other to implement scene rendering.

## Function Separation

Almost all WebGL engines, such as Three.JS, put major functions in browser. Different to these engines, our WebGL engine puts as more functions as possible on web server and as less functions as possible in browser.

The disadvantage of our function separation is that there are more interactions between browser and web server. To prevent too much interaction, we implement a buffer systems in our WebGL engine and caches as more data as possible in browser. If our WebGL engine can cache enough data in browsers, interactions in our WebGL engine can be as less as that in most WebGL engines, if all data is cached, it is the same as these engines implemented in browser. This is the reason why our WebGL employs such an architecture.

The first advantage of our function separation is flexible implementation. We can implement flexible function separations according to different scenes. Some functions are implemented in browser, some on web server, some functions even are implemented on both. Client-side function implementation can decrease communication and increase Interactivity and real-time ability. Server-side computers are usually of high performance. Our WebGL engine can gain advantages of both sides.

The second advantage of our function separation is scene data organization management and configuration. Client-side browser has no storage facility, all data are located on server-side web server that has file system. If all functions of a WebGL engine are implemented in browsers, user has to download scene data from web server, create and manage scene through engine API. If functions of a WebGL engine are implemented both in browsers and on web server, user can implement scene data organization management and configuration on server-side file system, only client-side needed data is necessary to be downloaded from web server.

The third advantage of our function separation is safety and security. Any function related to dangerous data can only be implemented on web server, client-side user can NOT touch dangerous data. In this way, data safety and security can be improved.

Function Separation in our WebGL engine can be illustrated as following figure.



Our WebGL engine implements most general functions in above figure, specific functions are implemented by an extensible driver architecture.

## Extensibility

WebGL is a shader-based APIs. Compare with fixed-function APIs, it allows more flexibility and sophisticated rendering techniques. Different graphics objects in a scene may rendered by different JavaScript programs and shaders.

Shader-based APIs requires WebGL engine architecture be more extensible. Different sever-side programs, different client-side programs, different shader programs, as well as different geometry data are easily and freely configured, i.e. easily added or deleted in WebGL engine.

Similar to those in operating systems, in our WebGL engine, extensible programs are also called **drivers**. There are driver types as following.

1. Server-side driver(Java language)
   1. Render driver: render corresponds to a set of programs. When configure system, you must identify class name of render driver and other programs (client-side JavaScript decode driver and draw driver, as well as shaders).
   2. Part driver: render plus geometry data creates part. Part corresponds to geometry object, i.e. shape object.
   3. Component driver: part plus position data creates component. Component corresponds to geometry objects at different position.
   4. Instance driver: multi-client may share same scene. instance is private object of a client for a component.
2. client-side driver(JavaScript language)
   1. Decode driver: JavaScript program that decode buffer object data from server. Usually this program creates frame data from mesh data.
   2. Draw driver: JavaScript program that employ GPU shader program and primitive data in buffer object to drawing scene. It is in this driver, we call WebGL API to render scenes.
3. GPU shader(OpenGL shader language)
   1. vertex shader
   2. fragment shader

## Developer Separation

Developer can be divided into four type as follow.

1. Model developer: Model developers are responsible for creating geometry model by software such as 3D Max, Maya, Solidworks. They need know nothing about program development, they need no knowledge about WebGL or rendering engine. They just need to know their model-build software and save their models into standard 3D file such as OBJ file, STL file, and .etc.
2. Web Front-end developer: they are web page developers. It is unnecessary for them to understand any computer graphics conception, such geometry, rendering, WebGL, etc. They just need to know <canvas> tags and call some simple JavaScript function for initialization and interaction in their webpage. Web Front-end developers need grasp simple JavaScript program development.
3. Scene configuring staff: edit scene configuration files on web server. They only need know how to configure a scene. Their software utility is text editor. No software development capability is needed. See document “Configuration” for more detail.
4. Driver developer: They are programmers for developing extensibility, including render driver, part driver, component driver, instance driver, as well as JavaScript driver. They also need develop shaders running on GPU. Driver developments are the most challenging job for programmers.
5. Engine maintainer: it is me, and I am waiting for your participation!

## Installation

1. Install java SDK and tomcat.
2. Copy engine java JAR file to tomcat webapps\ROOT\WEB-INF\lib directory. In this JAR file are all needed JAVA classes for you.
3. Copy engine web file to tomcat webapps\ROOT\. The file is a JSP file: interface.jsp. Client-programs in browser call this JSP file through AJAX for interaction between client and server.
4. Configure user data
   1. Put user data in a file directory.
   2. Create a temporary file directory.
   3. Configure user data: modify content in webapps\ROOT\ interface.jsp for identifying user data and temporary directory.
5. If necessary, develop new driver, copy server-side java program of the driver in webapps\ROOT\WEB-INF\class or webapps\ROOT\WEB-INF\lib, and configure new drivers in configuration file.
6. Done! Start tomcat. Browser your website!