

**ZED**

# **Engine Design Document**

**Open Game Developers**



# ZED

## Change Log

---

Version	Author	Changes
0.0.0.0	Rico	Initial revision



## Table of Contents

---

Introduction.....	1
Executive Summary.....	1
Overview.....	1
Design Goals.....	3
Proposed Engine Components.....	5
System.....	5
Arithmetic.....	5
Renderer.....	5
Input.....	6
Audio.....	6
Networking.....	6
Artificial Intelligence.....	6
Physics.....	6
Debug.....	6
Animation.....	6
Scene Management.....	6
Assets.....	6



# ZED

## Preface

---

This document describes the ZED Engine from a high-level perspective with no API-specific references or pre-conceived notions of what the engine is going to be doing at a low-level or at an external API-level.

General concepts, such as rendering viewports, network connections, or multi-channel speaker configurations will be described from a purely descriptive, non-specific viewpoint. To elaborate, no specific mention of the file descriptors for making network connections, or anything more detailed than the dimensions of a viewport, will be divulged.

Other than for the engine maintainers, this document will not be of much use. A Programmer's Guide document is planned for after the API is designed. For the design of the API, see the API Design Document. There are no programming samples in this document, the Programmer's Guide document, which is forthcoming, would be more appropriate for seeking information on how to use the API.





# ZED

## Introduction

---

### Executive Summary

ZED shall be designed to allow for developers to create interactive 3D worlds with a high-level and low-level API. Two paradigms are available; object-orientated and procedural, implemented in C++ and C, respectively. The decision to provide two APIs will give the developer a choice between a 'pure' C++ library and to allow for the developer to implement ZED on more processing-power limited hardware. For instance, the SEGA Saturn does not have a lot of room to allow for a class with multiple levels of inheritance and multiple abstract virtual functions; with a 4KiB cache, this is quite a hindrance. Adopting a C and C++ set of libraries enables developers to not have to compromise and attempt to create a similar interface if they want to use a ZED-like façade for their project.

### Overview

In making developers lives easier, ZED will be acting like an Operating System's microkernel or a true kernel. There will be core services available and from there modules are appended to allow for a minimal amount of dependencies. This will also allow for developers to extend the engine or use a different set of underlying APIs to implement an alternative library for a specific piece of hardware, for example.

Giving developers an API where they are in control of every path the code can take is an important goal of the engine. If the renderer is too rigid that it can only support Direct3D and OpenGL, then platforms that either implement a different API or cannot support them are instantly locked out of the development effort. This would lead to developers requiring a different engine, building their own, or trying to shoehorn in the rendering library they want to utilise. Using a modular approach to the implementation will enable and encourage a developer community to grow and contribute to the engine's advancement across a wide variety of hardware and platforms.

ZED will provide multiple libraries which will be supported by Open Game Developers. These libraries will encompass debugging, storage, memory management, exploiting processor capabilities, rendering, audio, networking, resource handling, and input handling. In addition to these libraries, there will be artificial intelligence, animation, and physics libraries as well as a stand-alone scripting language.



# ZED

## Design Goals

---

ZED should offer developers an easy-to-use, flexible, open, expandable, and modular engine.

It should be possible for developers to be able to get the engine's code and start writing games without having to attempt to accommodate the engine. The engine should accommodate the developer.

Keeping up-to-date documentation with a high degree of integrity is critical to new programmers who are evaluating the engine for integration.

Without too much effort, a new rendering technique or networking model should be easy to iterate on and integrate into the engine for other developers to use if they wish to distribute their code. Ideally, the new engine component should be able to integrate with the main development effort supported by Open Game Developers.

Wide variety of platform support:

### **Personal Computer Operating Systems**

Windows [2000, XP, 7]

Linux

Mac OS [10]

SunOS

### **Home and Handheld Consoles**

SEGA Saturn

SEGA Dreamcast

Microsoft Xbox

Nintendo DS

Nintendo GameCube

GP2X

Open Pandora

### **Tablets**

BlackBerry PlayBook



## Proposed Engine Components

---

### System

All other libraries use and expand upon this core functionality.

Basic hardware is accessed from the System API. This includes the memory, mass storage, and processor. External peripherals and any hardware components that are not critical to the operation of a computer (such as graphics cards and networking interfaces) are excluded from this library. Other hardware is accessed in their respective libraries.

In addition to providing interfaces to hardware; the API will be responsible for managing the hardware's resources, such as how to access memory and files. Making the best use of the processor requires knowing how many processors are available for parallelism and what the capabilities are.

### Arithmetic

Mathematical constructs which use the resources of the platform the application is being run on are vital to ensuring that 3D primitives are manipulated as fast as possible and that any intersection tests are performed in a timely fashion. For 3D primitives, there are vectors, matrices, and quaternions which will form the basis of manipulating 3D primitives. Intersections will be performed with rays, planes, bounding volumes, and polygons.

If there is a SIMD unit available on the processor (information provided by the System library) then It will be exploited to accelerate calculations.

### Renderer

Rendering is concerned only with the viewports being rendered into. Any windowing system specific activity is up to the user implementing their project. Fullscreen rendering and the like are not handled directly by ZED, only the rendering context is provided. Of course, some window system specific data types will need to be passed to the renderer so that rendering can occur.

The Renderer will need to scale from hardware on older platforms that doesn't support shaders, up to the latest-and-greatest graphics card that supports native GPGPU. As there are graphics cards with fixed function shaders, there will need to be a fallback for those cards. For programmable function shaders, there will be support for applying and managing them as well as interfaces to set the data they require. In both the programmable and fixed function cases, vertices and textures will need to be managed. The renderer does not have any concept of scene management, animation of any sort, nor models as comprised from meshes. These will be supported from separate components.

Shaders are implemented on a per-graphics capability set, meaning that the user will have to specify the minimum graphics profile available and ensure they offer valid alternative code execution paths for differing profiles. Built-in shaders are provided for rendering lines and basic textured, lit vertices.

Textures will utilise file formats that the graphics device natively supports, such as S3TC, PVRTC, ATITC, ETC, and DXTx/BCx. Online conversion of textures would be ideal for testing textures.

### Input

There is a myriad of hardware available for input today and there will be many more input devices in the future. To help abstract the differences in hardware, the input library provides a way to link input actions to a common processable format for games which is defined by the user. As hardware will be vastly different, there should be a common interface for accessing the input device information required by a title.

### Audio

Playing compressed and uncompressed audio through multi-channel audio devices. In a nutshell.

### Networking

Provides functions to connect remote machines, whether they be on a LAN, MAN, or WAN. Sending and receiving data, and packing and unpacking aforementioned data. Client/Server and Peer-to-Peer connectivity paradigms.

### Artificial Intelligence

This library is responsible for imbuing non-player characters with seemingly intelligible actions; steering behaviours, pathfinding, representations of the world for the agent to navigate, state machines, and behaviour trees. All of the techniques should be open for users to provide values that change the behaviour of the agent. Blending of different states and behaviours should be possible.

### Physics

Rigid-body and soft-body physics as well as collision detection will be handled by this library. Mass-spring and particle physics will be employed.

### Debug

Debugging information can come in a variety of forms. Whether it's showing normals of a model or a command-line console, debugging functions are imperative to tracking down errors in the execution of a game. The Debug library will have to draw to the screen as well as communicate over a network for remote debugging.

### Animation

While animation is related to rendering, it is not included in the Renderer library because it is another layer building upon the core rendering functions. Morphing, skeletal, and per-vertex animation will be supported.

### Scene Management

Scene management is intended to be flexible and extensible for other systems of world partitioning systems such as BSP trees, Octrees, Quadrees, and PVS. Vertex animation is not the specific per-vertex animation system, rather it is the broad spectrum of per-vertex, skeletal, and morphing animation systems.

### Assets

Vertices can be rendered as specified procedurally or from a ZED model file. Model files can be static models or animated with skeletal animation, shaders and textures are also referenced in model files.