

Die 3D-Engine Ogre - Made in Guernsey

Ogre steht für „Object-Oriented Graphics Rendering Engine“ und ist eine Szene-orientierte 3D-Engine. Wer das Ogre-Logo betrachtet, kann auch eine zweite Bedeutung des Begriffs ableiten. Das Wort Ogre heißt im Englischen sinngemäß Menschen-fressendes Ungeheuer.



Ogre ist ein Open-Source-Projekt. Der öffentlich zugängliche Quellcode steht unter der LGPL-Lizenz (GNU Lesser General Public License). [Steve Streeting](#)[∞] (Nickname: sinbad) hat die Arbeit an Ogre im Jahr 2001 begonnen. Er steuert von Guernsey aus das internationale Ogre-Entwicklerteam. Die engsten Mitarbeiter sind Jianhua Xie (genva) aus Peking und Phillip J. Castaneda (pjast) aus Kalifornien. Ogre ist eine sehr komplexe und mächtige Grafiksoftware, die sehr intensiv weiterentwickelt wird. Mehr als 8.000 Mitglieder sind im offiziellen Forum von [ogre3d.org](#) registriert, und die sehr aktive und hilfsbereite Community unterstützt Einsteiger wie Profis. In Deutschland ist das [ogre3d.de](#)-Forum erst seit Mai 2006 online.

Im Februar 2005 wurde das Final Release v1.0.0 veröffentlicht. Seit dem 7. Mai 2006 ist mit Ogre v1.2 die neue Version Dagon erhältlich.

Zu den herausragenden Eigenschaften von Ogre gehört das elegante Designkonzept. Der Ogre-Quellcode nutzt die modernen Elemente von C++ konsequent. Wer mit Begriffen wie Namespaces, Templates, Design Pattern, STL oder virtuellen Funktionen gar nichts anfangen kann, sollte vor dem Start der Arbeit mit Ogre seine C++ Kenntnisse auffrischen.

Ogre ist als Grafikengine auf den Bereich von Real-Time-Rendering generell spezialisiert und nicht als Game Engine angelegt. Vorteil: Die Engine kann flexibel in jeglicher Art von Anwendung zum Einsatz kommen. Die Bereiche Sound, Input, Physik, Künstliche Intelligenz und anderes mehr muss der Entwickler allerdings grundsätzlich selbst einbinden, wenn er sie nutzen will. Das bietet die Freiheit, jeweils die passende und leistungsfähigste Kombination zu wählen. Flexibel ist die Engine auch, wenn es um die zugrundeliegende 3D-API (Application Programming Interface) geht.

Ogre beherrscht Direct3D und OpenGL

Die für den Ogre-Entwickler sichtbaren Elemente der Klassenbibliothek abstrahieren von der Betriebssystem-Plattform und damit auch von der anzusprechenden Grafikschnittstelle. Für die Arbeit mit Ogre sind daher auch keine Direct3D- oder OpenGL-spezifischen Programmierkenntnisse notwendig, allerdings helfen diese sicherlich sehr beim Verständnis der Vorgänge, die von Ogre in Gang gesetzt werden. Die direkte Ansprache der API, über die gerendert werden soll, erfolgt ausschließlich über Interface-Klassen und läuft quasi im Hintergrund ab. Ogre ist damit Plattform-unabhängig und unterstützt Direct3D genauso wie OpenGL.

Jede Ogre-Anwendung spielt sich im Namensraum (namespace) Ogre ab. Dies stellt sicher, dass keine Überschneidungen mit Bezeichnern stattfinden, die im eigenen Quellcode oder in

eingebundenen Bibliotheken benutzt werden. Ogre ist von der Design-Philosophie her so gestaltet, dass die Engine ideal innerhalb anderer Anwendungen zum Einsatz kommen kann. Der Zugriff auf die Objekte erfolgt durch die Voranstellung von: Ogre::

Beispiele für den Zugriff auf Ogre-Objekte:

Ogre::Camera

Ogre::SceneManager

Ogre::Plane

Ogre::Animation

Standardkonform kann auf das vorangestellte Ogre:: verzichtet werden, wenn die using-Direktive zum Einsatz kommt: using namespace Ogre;

Die Wurzelklasse Root

Ein Ogre-Programm startet immer über die Klasse Ogre::Root. Von diesem Einstiegspunkt aus wird alles Weitere abgeleitet. Root muss als erstes Objekt initialisiert werden und als letztes aus dem Speicher wieder verschwinden. Das Root-Objekt verbindet die Ogre-Anwendung mit der Systemumgebung. Von hier aus werden die verfügbaren Render-Systeme aktiviert, es werden die Konfigurationen verwaltet und der Zugriff auf sämtliche anderen Klassen des Systems gesteuert. Root ist die Wurzel der Ogre-Anwendung, von der alle anderen Objekte abhängen und erreichbar sind. Sobald eine Instanz des Root-Objektes geschaffen wurde, ist es über die Laufzeit der Anwendung auf zwei Wegen ansprechbar: als Referenz oder über einen Pointer (Zeiger).

ogre.cfg

```
Root::getSingleton    // Referenz
Root::getSingletonPtr // Pointer

Der Konstruktor für Root sieht wie folgt aus:

Ogre::Root::Root (const String &pluginFileName = "Plugins.cfg",
                 const String &configFileName = "ogre.cfg",
                 const String &logFileName = "Ogre.log"
                 )
```

Von den drei Dateien plugins.cfg, ogre.cfg und Ogre.log muss die erste Datei beim Programmstart bereits existieren. ogre.cfg wird benutzt, wenn sie existiert, und Ogre.log ist ein klassisches Logfile.

Das Konfigurationsfile plugins.cfg enthält Informationen über die Ogre-Plugins und ihren Platz auf dem Rechner. Diese Datei kann in der Regel ohne Änderung des Inhalts für verschiedene eigene Ogre-Anwendungen genutzt werden.

Die Konfigurationsdatei für die Rendersysteme sieht beispielsweise wie folgt aus und ist im OgreSDK/bin/debug-Verzeichnis zu finden. Die ersten zwei Angaben aus diesem File finden sich bei den mitgelieferten Beispielanwendungen (ExampleApplication) von Ogre im Abfragedialog vor dem Programmstart wieder. Die Rendersysteme sind als abstrakte Klassen konzipiert, die ein Interface

zur darunter liegenden 3D-API bilden.

plugins.cfg

```
# Defines plugins to load

# Define plugin folder
PluginFolder=.

# Define plugins
Plugin=RenderSystem_Direct3D9
Plugin=RenderSystem_GL
Plugin=Plugin_ParticleFX
Plugin=Plugin_BSPSceneManager
Plugin=Plugin_OctreeSceneManager
Plugin=Plugin_CgProgramManager
```

Die zweite Konfigurationsdatei, ogre.cfg, ist für jede Ogre-Anwendung einmal vorhanden und enthält detaillierte Angaben über die Render Systeme. Für die mitgelieferten Beispielanwendungen enthält die Datei die folgenden voreingestellten Festlegungen.

ogre.cfg

```
Render System=Direct3D9 Rendering Subsystem

[Direct3D9 Rendering Subsystem]
Allow NVPerfHUD=No
Anti aliasing=None
Floating-point mode=Fastest
Full Screen=Yes
Rendering Device=ASUS RADEON A9200SE
VSync=No
Video Mode=800 x 600 @ 32-bit colour

[OpenGL Rendering Subsystem]
Colour Depth=32
Display Frequency=100
FSAA=0
Full Screen=Yes
RTT Preferred Mode=FBO
VSync=No
Video Mode=1024 x 768
```

Sobald das Root-Objekt initialisiert ist, können über Pointer die weiteren Objekte des Systems angesprochen werden. Das Root-Objekt stellt die Methode startRendering zur Verfügung. Wird diese aufgerufen, dann rendert der Rechner in einer kontinuierlichen Abfolge (continuous rendering

loop). Diese Ereignisschleife wird erst dann unterbrochen, wenn entweder alle Fenster geschlossen werden oder wenn ein FrameListener-Objekt den Zyklus unterbricht. FrameListener-Objekte gehören zu den zentralen Werkzeugen eines Ogre-Entwicklers. Sie werden in einem späteren Tutorial eingehend beschrieben.

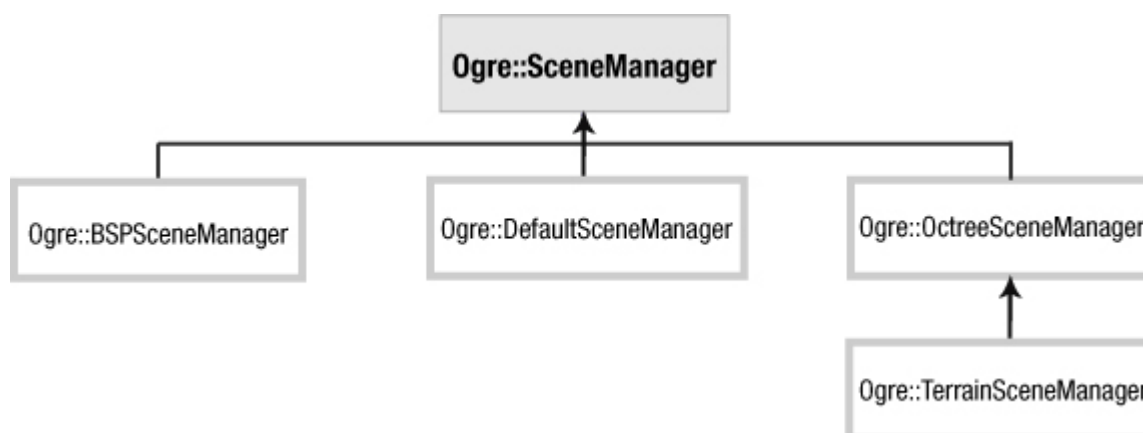
Für den Aufbau einer Ogre-Anwendung gehören Root-Objekt und RenderSystem zu den Kernelementen. Beide sind lebenswichtig. Allerdings werden diese Klassen im Normalfall nicht direkt angesprochen. Es ist wichtig zu wissen, dass es sie gibt. Viel stärkere Aufmerksamkeit verdienen SceneManager und SceneNodes, die das lebendige Skelett jeder Ogre-Anwendung darstellen.

Manager, Knoten und Wesen

SceneManager, SceneNode und Entity heißen die drei wichtigsten Konstruktionselemente, mit denen jede Ogre-Anwendung arbeitet. Für das Verständnis der Szene-orientierten Engine Ogre ist das Wissen über den grundsätzlichen Aufbau und das Zusammenspiel dieser Klassen entscheidend.

Die SceneManager-Klasse ist für alles verantwortlich, was sich innerhalb einer Szene abspielt, sie verwaltet die Organisation und das Rendern einer Szene. Mit Szene ist dabei die 3D-Umgebung gemeint, die viele Elemente enthalten kann: 3D-Objekte (Mesh), Nebel (Fog), Lichtquelle (Camera), Himmel (Skydome) und vieles mehr. Alles was auf dem Bildschirm erscheint wird von einem SceneManager verwaltet. Über ihn werden sämtliche Inhalte und sämtliche Positionsdaten für die Inhalte gesteuert. Der SceneManager verfügt zudem über alle Informationen im Blick auf das Material und die Methoden jeglicher Objekte der Scene. Vom SceneManager aus erfolgt die Nachricht an das RenderSystem-Objekt, sobald der Zeitpunkt für das Rendern der Scene gekommen ist. Der Methodenaufruf SceneManager::_renderScene muss nicht direkt gestartet werden, sondern er wird immer dann automatisch aufgerufen, wenn das zu rendernde Element im Programmablauf neu aufgebaut wird.

Es stehen vier SceneManager zur Verfügung. Voreingestellt kommt die Klasse Ogre::DefaultSceneManager zum Einsatz, die als Allzweckwaffe den größten Einsatzbereich abdeckt.



Die drei Klassen BSPSceneManager, OctreeSceneManager und TerrainScene Manager sind jeweils Spezialisten für bestimmte Weltgeometrien. Der BSPSceneManager ist auf die Darstellung von Umgebungen spezialisiert, die Innenräume abbilden. BSP steht für Binary Space Partition und ist ein spezielles rekursives Baumstruktur-Verfahren, mit dem die Scene gerendert wird. Der

OctreeSceneManager rendert nach einem anderen Algorithmus, bei dem die baumförmige Datenstruktur Octree zugrunde liegt. Von diesem abgeleitet ist der auf Landschaftsdarstellungen spezialisierte TerrainSceneManager. Bei diesem kann ein Terrain inklusive Angaben über Texturen, Größenverhältnisse und anderes mehr über eine Konfigurationsdatei (zum Beispiel terrain.cfg) geladen werden.

In einer Ogre-Anwendung können mehrere SceneManager zum Einsatz kommen, die nebeneinander existieren und gemeinsam die Szene aufbauen.

Ogre unterscheidet zwischen den vielen kleinen und größeren beweglichen Objekten in der Welt und der generellen Szeneriegestaltung, die großen Raum einnimmt und fest positioniert ist (Level-Geometrie).

Eine Entity (Weseneinheit) definiert die Instanz eines Objektes, das als Meshmodell aufgebaut ist und keine fixe Position in der Weltgeometrie einnimmt. Mesh-Objekte können alle Formen annehmen, von Monstern, Menschen, Orks und Elfen bis hin zu Beluga-Walen. Weitere bewegliche Weltelemente wie Lichtquellen (Camera), Partikel oder Nebel gelten nicht als Entity, sie werden direkt von Ogre aus generiert.

Die Mesh-Objekte können als Dateien mit der Endung .mesh geladen werden. Sie stammen zumeist aus externen 3D-Modellierungsprogrammen wie beispielsweise 3D Studio Max, Maya oder Blender und werden von dort mittels eines Hilfsprogramms (Exporter) in das .mesh-Format umgewandelt.

Entities und damit Mesh-Objekte können auch innerhalb von Ogre selbst von Hand (manual) gefertigt werden. Dazu dient die folgende Methode: MeshManager::createManual

Jedes zu rendernde Objekt wird in Ogre getrennt von seinen Positionsdaten verwaltet. Das bedeutet, dass kein Objekt direkt innerhalb der Scene an einer bestimmten Position platziert wird. Jedes Mesh und damit jede Entity muss an ein SceneNode-Objekt andockt werden, da dieses die Positions- und Orientierungsdaten aller Objekte verwaltet. Die SceneNodes wiederum müssen dem SceneManager bekannt gemacht werden und werden von ihm abgeleitet. SceneNodes sind genauso wenig sichtbare Elemente wie SceneManager, sie bekommen erst dann eine Repräsentation auf dem Bildschirm, wenn ihnen Entities oder andere Objekte wie Lichtquellen, Nebel etc. zugeordnet werden. Einem SceneNode können beliebig viele Entities zugeordnet werden, und es können Hierarchien von SceneNodes konstruiert werden. Die Position von abgeleiteten SceneNodes (ChildSceneNodes) ist immer nur relativ zu dem Elternknoten definiert. Das heißt, dass sich die zugeordneten Objekte, die in der Hierarchie tiefer stehen, an den höheren SceneNodes orientieren. Soll beispielsweise ein Taucher dargestellt werden, der von einem Ungeheuer mit Leuchtaugen begleitet wird, würde zunächst der Taucher an den SceneNode andockt werden. Dann würden das Ungeheuer und die Lichtquellen als Objekte ebenfalls an den SceneNode gemeldet. Wird nun die Position des SceneNodes verändert, bewegt sich mit dem Taucher auch das Glühmonster.

Über die Klassen SceneManager, SceneNode und Entity wird sehr viel in der Ogre-Welt gesteuert. Es sind sehr mächtige Klassen. In diesem Tutorial wurde nur ein kleiner Ausblick auf die Leistungsfähigkeit dieser Objekte gegeben. Jetzt können die ersten Experimente in einer eigenen Anwendung folgen. Die Inhalte dazu liefert das Ogre-SDK gleich mit, denn es enthält einige fertige Modelle zum Testen.

[CategoryOgre3d](#)