Framework

Einleitung in das Framework von Orxonox

Slides by
Fabian Landau

Was gehört alles zu einem Framework?

Grafik

Physik

Speicherverwaltung

Resourcen

Network

Sound

Input

Speichern/Laden

Konfiguration

Log Output

Zeitmanagement

Reflection

Objektverwaltung

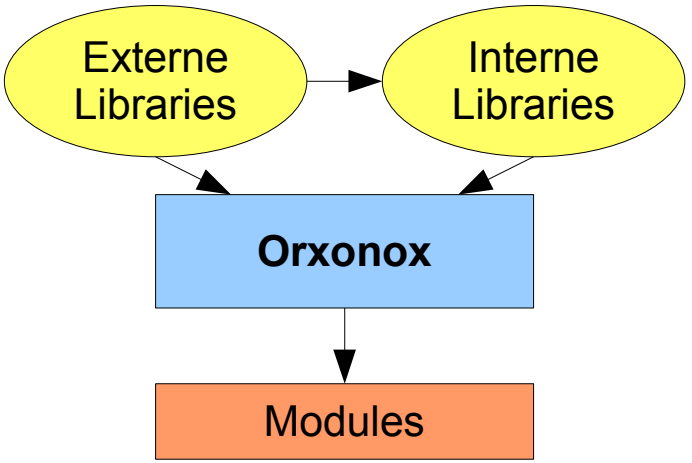
Framework

Ablauf dieser Einführung:

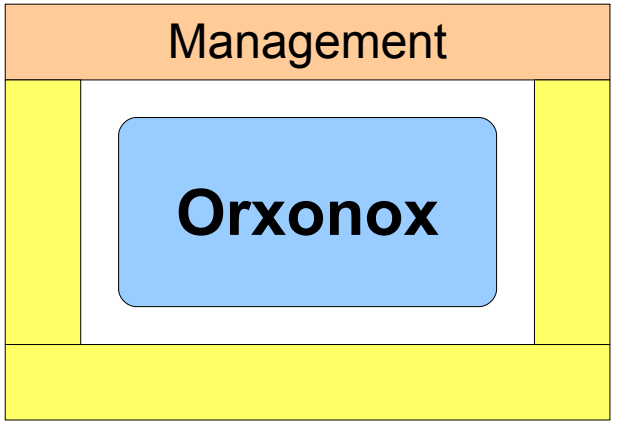
⇨ • Framework: Definition und Aufgaben
• Externe Libraries: Ogre, CEGui, Bullet
• Klassenhierarchie
• Worldentities
• Grafische Komponenten
• Beispielklasse
• XML

Framework

Was ist ein Framework?
Was sind die Aufgaben des Frameworks?

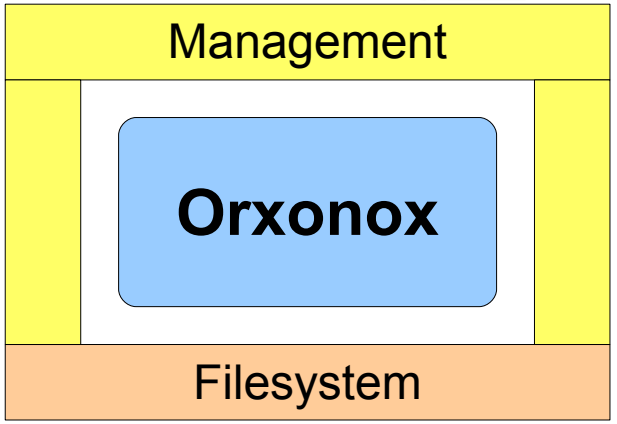


Framework



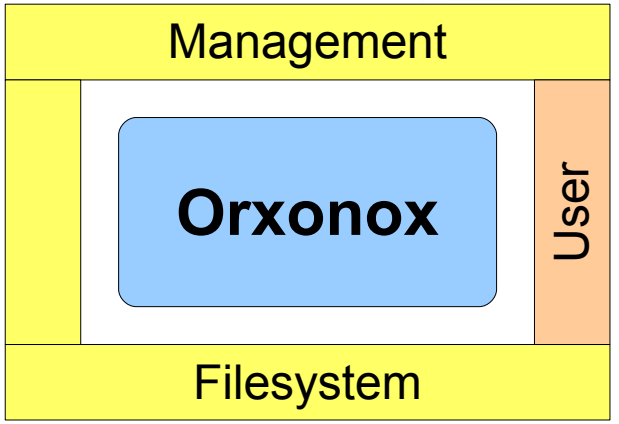
- Verwaltung aller Objekte die existieren
- Bereitstellung von Hilfsfunktionen und -klassen (z.B. für Stringmanipulationen oder Typeconversions)
- Ablauf des Main-Loops (ein Durchlauf entspricht einem Frame)
- Interface zu externen Libraries

Framework



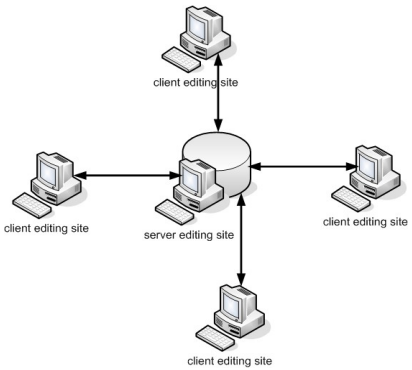
- Speichern von Logs
- Konfiguration von Parametern und Speichern von Einstellungen in Config-Files
- Lesen und Schreiben von XML-Files für Levels und andere Zwecke
- Zugriff auf Resourcen (Models, Texturen, Shader, usw.)

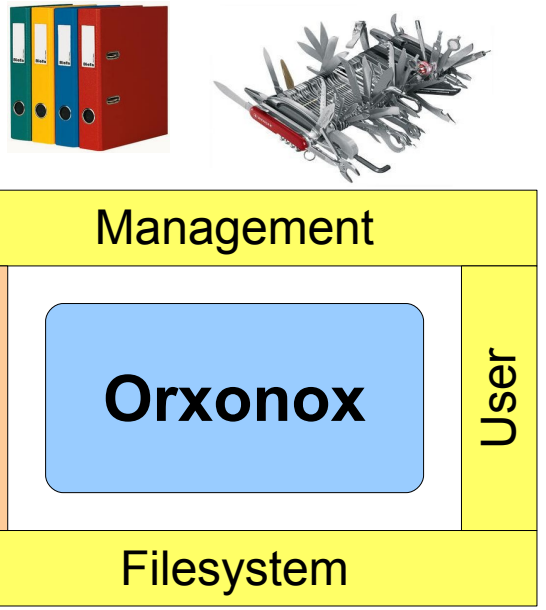
Framework



- Erzeugen einer grafischen Darstellung von zwei- und dreidimensionalen Elementen (Grafikengine)
- Verwaltung der Position und Bewegung aller physikalischer Objekte im Raum (Physikengine)
- Ausgabe von Musik und Soundeffekten (Soundengine)
- Verarbeitung von Input über Tastatur/Maus/Joystick (Inputengine)

Framework





- Server: Berechnen der Gamelogik
- Server: Synchronisieren aller Clients
- Client: Darstellen der Objekte
- Client: Vorverarbeiten und Senden von Userinput an den Server

Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- ⇨ Externe Libraries: Ogre, CEGui, Bullet
- Klassenhierarchie
- Worldentities
- Grafische Komponenten
- Beispielklasse
- XML

Framework

Ogre (Grafikengine)

Framework

Ogre (Grafikengine)

Framework

Ogre (Grafikengine)

Framework

Ogre (Grafikengine)

Framework

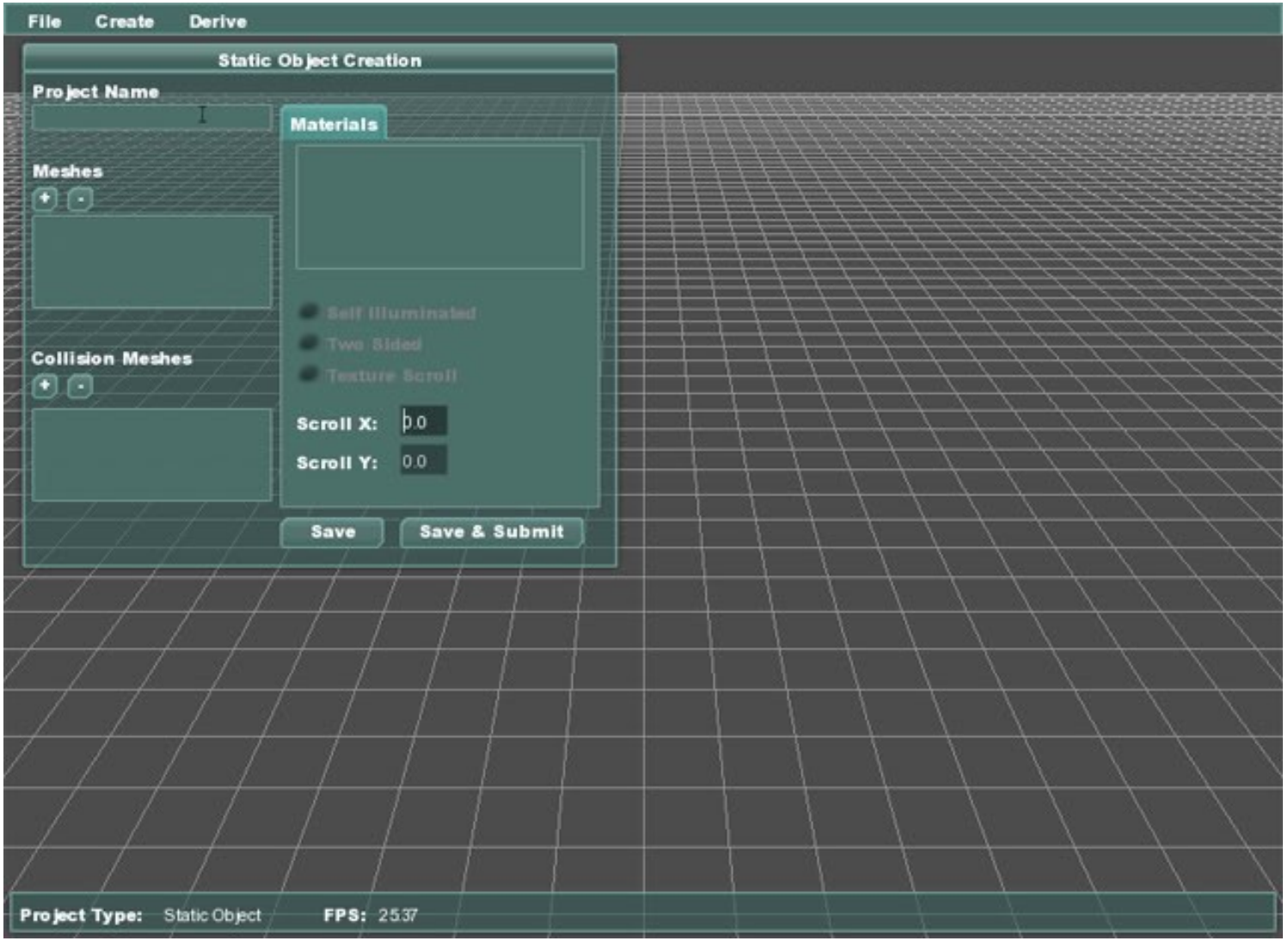
Ogre (Grafikengine)

Framework

CEGui (GUI-Engine)

Framework

CEGui (GUI-Engine)

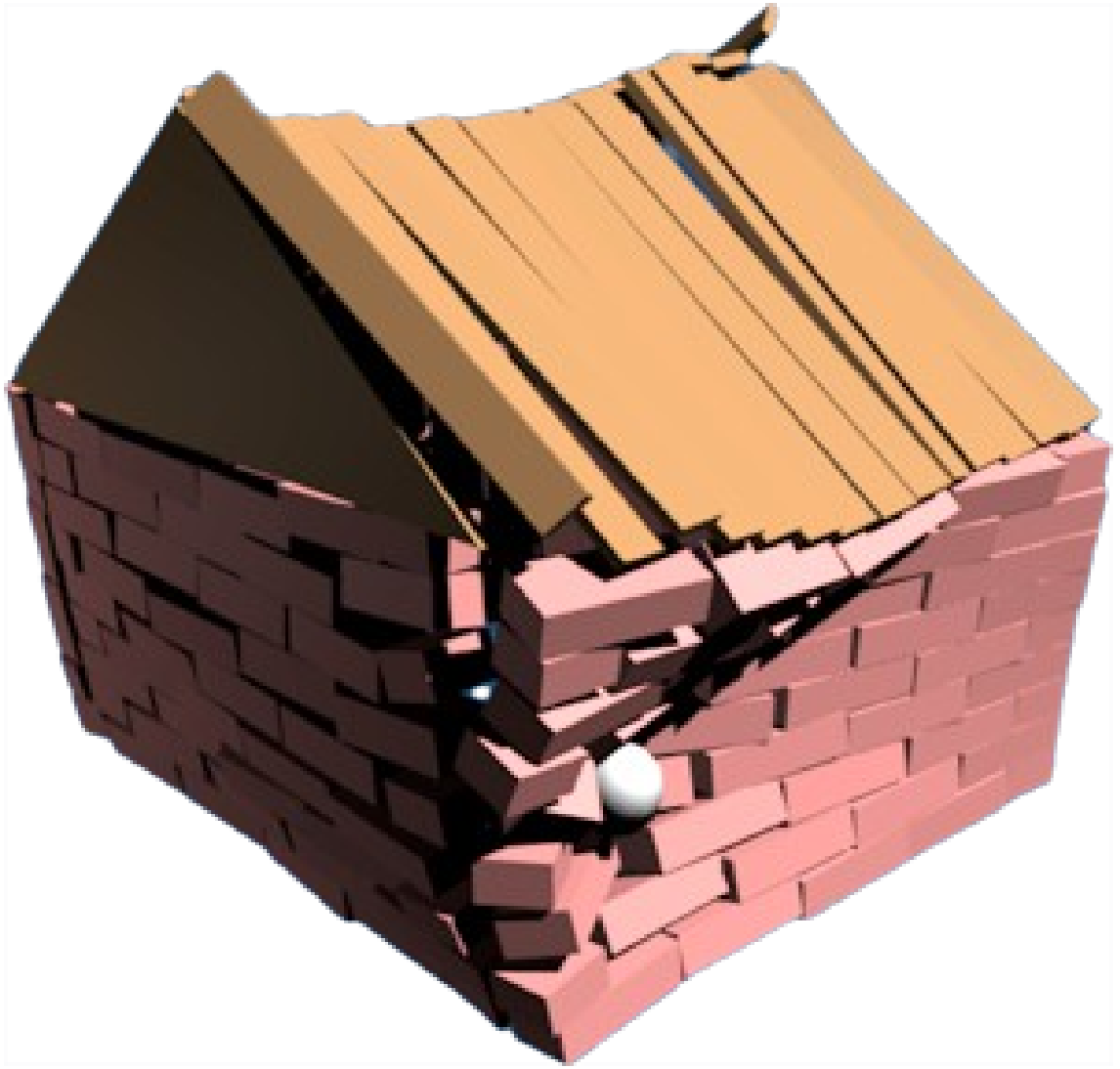


Framework

CEGui (GUI-Engine)

Framework

Bullet (Physikengine)



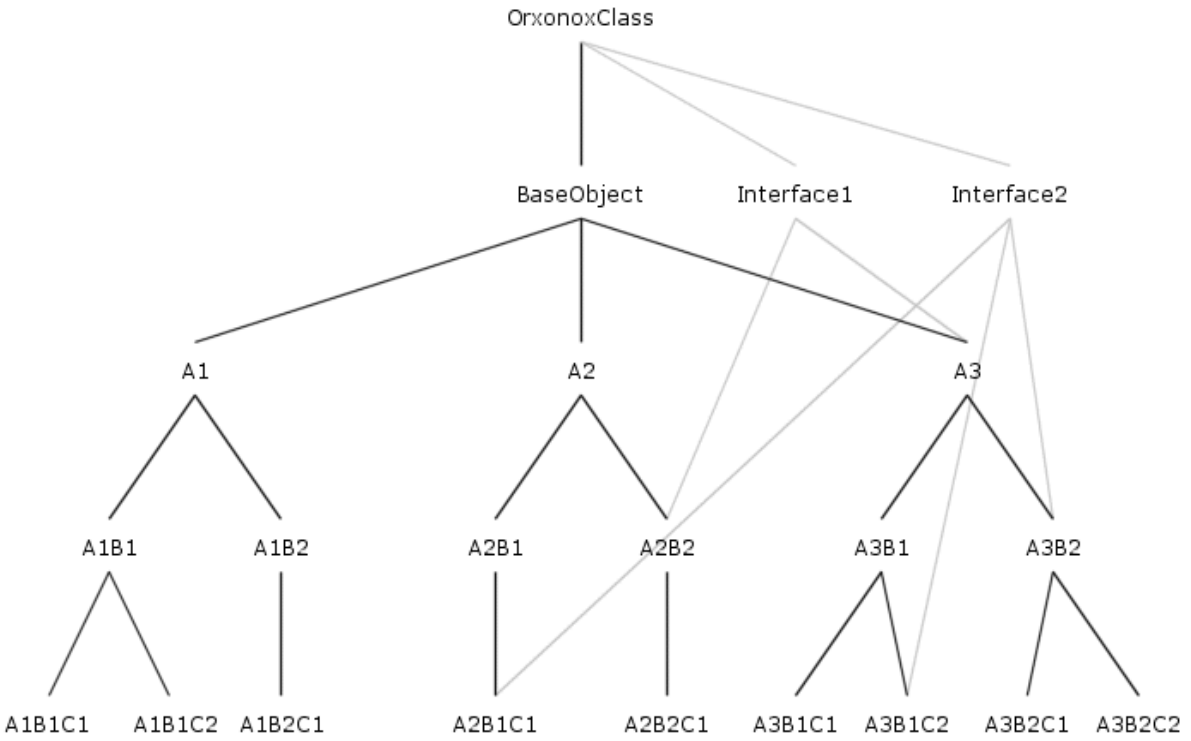
Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- Externe Libraries: Ogre, CEGui, Bullet
- ⇨ Klassenhierarchie
- Worldentities
- Grafische Komponenten
- Beispielklasse
- XML

Framework

Klassenhierarchie



Framework

OrxonoxClass

- OrxonoxClass ist die Basisklasse aller Klassen in Orxonox, inklusive Interfaces und abstrakten Klassen.
- Die OrxonoxClass enthält Funktionen und Variabeln, die für das Framework notwendig sind, nicht aber für den normalen Gebraucht der Klasse.

Framework

BaseObject

- BaseObject ist die Basisklasse aller Objects in Orxonox.
- Als „Objects“ bezeichnen wir Klassen, die eine direkte oder indirekte Wirkung auf das Spiel haben, oder aber (direkt oder indirekt) vom Spiel beeinflusst werden.
- Vereinfacht ausgedrückt bezeichnet man mit „Object“ alle jene Klassen, deren Instanzen durch ein Levelfile geladen werden und am Ende eines Levels wieder zerstört werden.

Framework

Interfaces

- Ein Interface in Orxonox kann, im Gegensatz zu Java, Code enthalten. Allerdings kann man es nicht erzeugen.
- Interfaces werden Verwendet, um Gemeinsamkeiten von Klassen, die nicht direkt miteinander verwandt sind, zu vereinen.

Framework

Objekthierarchie: Tickable

- Tickable ist ein Interface.
- Klassen die von Tickable erben, erben die Funktion `tick(float dt)`
- Die Funktion `tick(float dt)` wird in jedem Frame einmal aufgerufen, wobei der Wert von `dt` die Zeit seit dem letzten Aufruf in Sekunden angibt.
- Von Tickable zu erben und `tick(float dt)` zu implementieren ist die einzige Möglichkeit, ein aktives Objekt zu erzeugen (mit Ausnahme von Timern). Objekte ohne Tick bleiben das ganze Spiel über statisch.

Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- Externe Libraries: Ogre, CEGui, Bullet
- Klassenhierarchie
- ⇨ Worldentities
- Grafische Komponenten
- Beispielklasse
- XML

Framework

Worldentities: Definition

- WorldEntity ist die Basisklasse aller Objekte, die eine Position im Raum haben.
- Wir unterscheiden zwischen statischen, beweglichen und kontrollierten Worldentities. Die Unterscheidung geschieht eigentlich lediglich aus Performancegründen (Physik und Network), lohnt sich aber definitiv.
    - Statisch: StaticEntity (z.B. eine Spacestation)
    - Beweglich: MovableEntity (z.B. ein Projektil)
    - Kontrolliert: ControllableEntity (z.B. ein Spaceship)

Framework

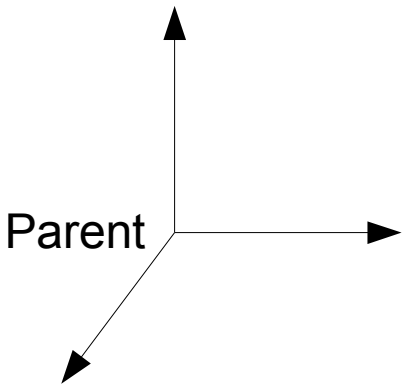
Worldentities: Definition

- Worldentities können aneinander „attached“ werden, d.h. man kann kann sie zusammenhängen. Die Position des angehängten Objekts (Child) ist dann relativ zur Position und Rotation des Basisobjekts (Parent).

Framework

Worldentities: Attachen

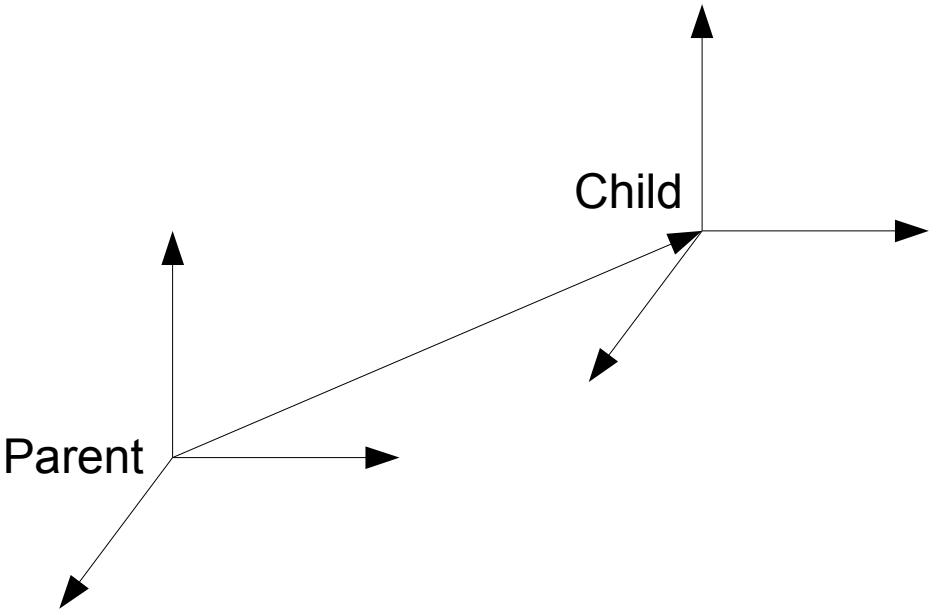
- Absolute Position im Raum (Parent):



Framework

Worldentities: Attachen

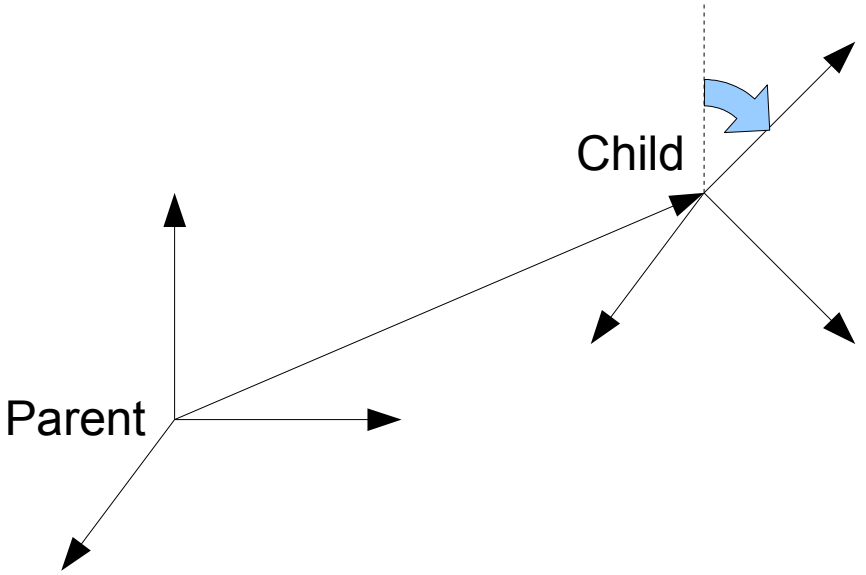
- Relative Position im Raum (Child):



Framework

Worldentities: Attachen

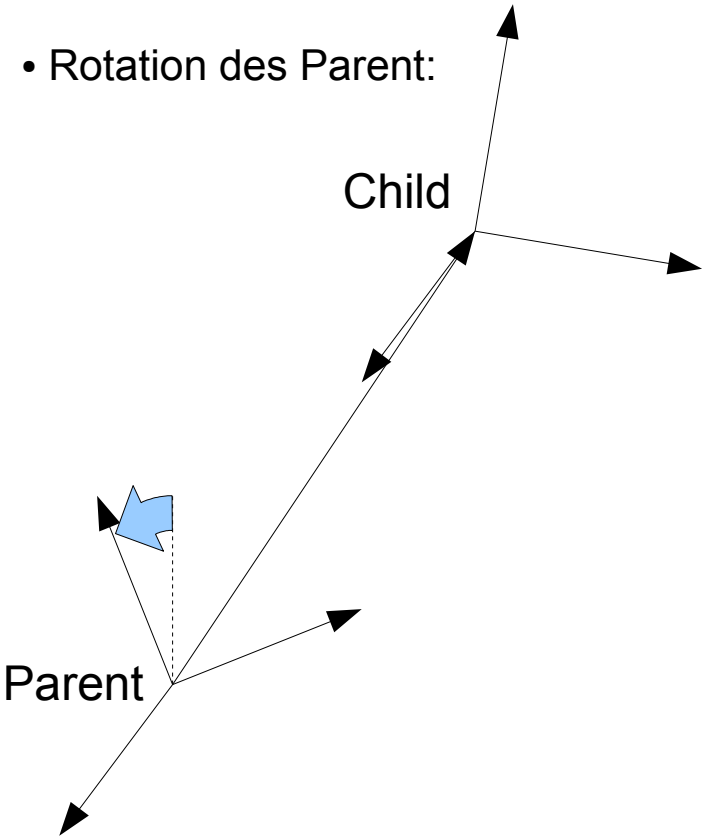
- Rotation des Child:



Framework

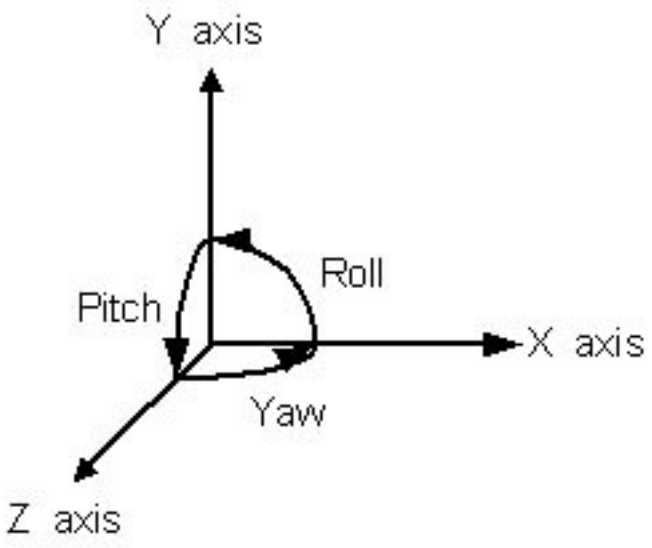
Worldentities: Attachen

- Rotation des Parent:



Framework

Worldentities: Koordinatensystem



- X und Y-Achse beschreiben den zweidimensionalen Bildschirm.
- Die Z-Achse muss aus dem Bildschirm herausragen, um ein Rechtskoordinatensystem zu bilden (Rechte-Hand-Regel).
- „Vorwärts“ ist also die negative Z-Achse, d.h. (0, 0, -1)
- Die Rotation erfolgt ebenfalls gemäss einer Rechte-Hand-Regel: Daumen in Richtung der Achse, Rotation in Richtung der restlichen Finger (wie bei der Elektromagnetischen Induktion).

Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- Externe Libraries: Ogre, CEGui, Bullet
- Klassenhierarchie
- Worldentities
- ⇨ Grafische Komponenten
- Beispielklasse
- XML

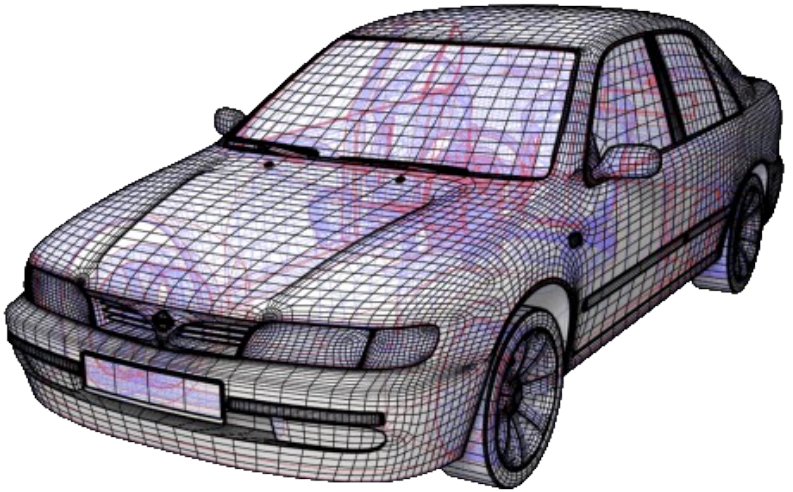
Framework

Grafische Komponenten: Prinzip

- Nur Objekte mit einer Position im Raum können eine grafische Representation haben (ausgenommen 2D-Elemente).
- Um beim Designen unserer Objekte flexibel zu sein, verwenden wir für grafische Komponenten eine HasA Relation anstelle einer IsA Relation.
- Das bedeutet, dass z.B. ein Spaceship nicht von der Klasse „Model“ erbt, wenn es ein Model haben will, sondern dass wir ein Model ans Spaceship attachen.
- Aus diesem Grund sind alle grafischen Komponenten Worldentities.

Framework

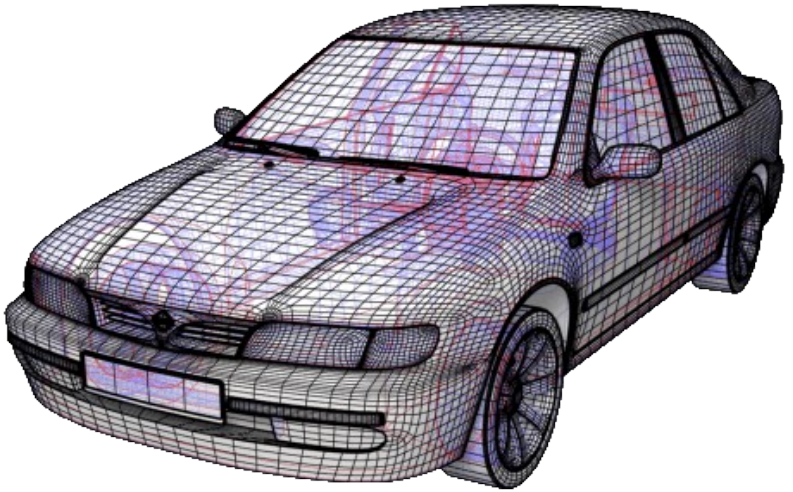
Grafische Komponenten: Model



- Ein Model ist ein dreidimensionales, texturiertes Objekt, das in Blender erzeugt wurde und mit dem Ogre-Mesh-Exporter in ein Mesh konvertiert wurde.

Framework

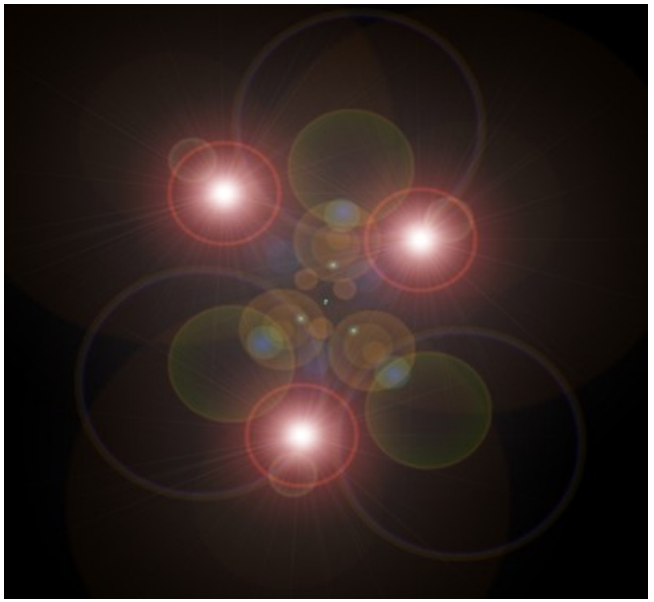
Grafische Komponenten: Model



- Die Texturen werden als Materials gespeichert. Materials enthalten einen Link auf die Textur-Datei, sowie zusätzliche Informationen wie z.B. Beleuchtungs- und Reflektionsparameter oder Bumpmaps und sonstige Shader.

Framework

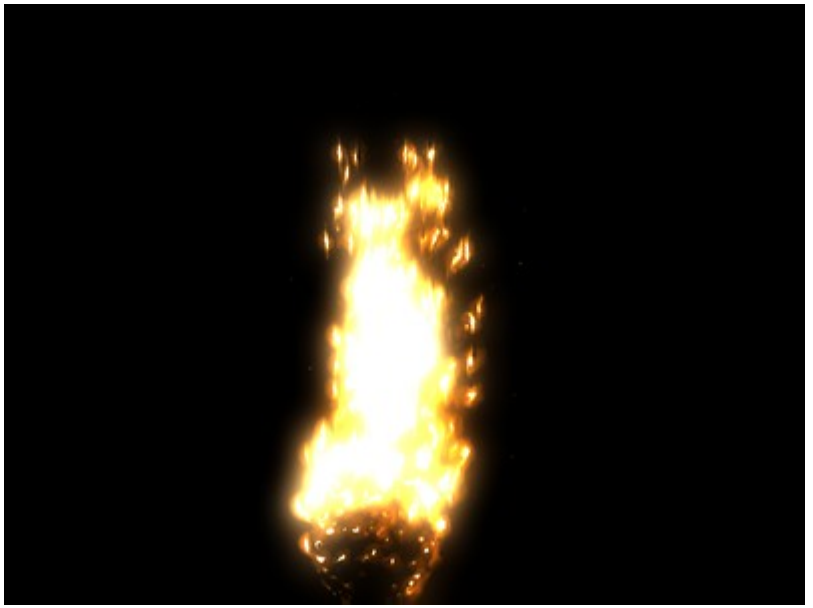
Grafische Komponenten: Billboard



- Billboards sind zweidimensionale Texturen die immer zur Kamera ausgerichtet sind.

Framework

Grafische Komponenten: ParticleEmitter



- Ein ParticleEmitter erzeugt einen dauerhaften Partikeleffekt, z.B. Rauch oder Feuer.
- Ein ParticleSpawner ist ebenfalls ein ParticleEmitter, allerdings zerstört er sich nach kurzer Zeit. Er eignet sich daher für kurze Partikeleffekte wie z.B. Explosionen oder Mündungsfeuer.

Framework

Grafische Komponenten: Light

- Mindestens eine Lichtquelle wird benötigt, damit die Szene nicht komplett schwarz bleibt.
- Man unterscheidet drei Typen Licht:
  - Ambient Light (beleuchtet alles gleichmässig, ohne Richtung, ohne Schattenwurf, ohne Gradient).
  - Point Light (geht von einem Punkt aus, wirft Schatten, wird auf Entfernung schwächer)
  - Spotlight (wie Point Light, allerdings wirft es lediglich einen Lichtkegel)

Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- Externe Libraries: Ogre, CEGui, Bullet
- Klassenhierarchie
- Worldentities
- Grafische Komponenten
- ⇨ Beispielklasse
- XML

Framework

Beispielklasse: CMakeLists.txt

- Wir erstellen zwei neue Dateien, MyClass.cc (das Source-File) sowie MyClass.h (das Header-File).
- Im gleichen Ordner in dem wir die Files erstellt haben, suchen wir die Datei „CMakeLists.txt“ und suchen nach einer Liste von anderen Source-Files. Dort Tragen wir MyClass.cc an einer beliebigen Stelle ein.
- Dadurch wird sichergestellt, dass unser neues File kompiliert wird.

Framework

Beispielklasse: Header

- Im Header-File Deklarieren wir die neue Klasse:

```
class MyClass : public MovableEntity
{
     public:
          MyClass(BaseObject* creator);
          virtual ~MyClass();



          virtual void tick(float dt);
};
```

- Unsere Klasse erbt also von MovableEntity (ein bewegliches WorldEntity).
- Da MovableEntity ausserdem vom Interface Tickable erbt, erbt auch unsere Klasse die Tick-Funktion.

Framework

Beispielklasse: Source

- Im Source-File Implementieren wir das Grundgerüst der neuen Klasse:

```
MyClass::MyClass(BaseObject* creator)
{
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
}
```

Framework

Beispielklasse: CreateFactory

- Zuerst müssen wir eine Factory erstellen, damit unsere Klasse vom Framework erkannt und auch über XML geladen werden kann:

```
CreateFactory(MyClass);

MyClass::MyClass(BaseObject* creator)
{
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
}
```

Framework

Beispielklasse: RegisterObject

- Als nächstes müssen wir direkt zu Beginn des Constructors unser Objekt registrieren:

```
CreateFactory(MyClass);

MyClass::MyClass(BaseObject* creator)
{
     RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
}
```

Framework

Beispielklasse: Creator

- Ausserdem müssen wir den creator-Pointer an die Basisklasse weitergeben. Er übermittelt den Kontext, in dem ein Objekt erzeugt wurde:

```
CreateFactory(MyClass);

MyClass::MyClass(BaseObject* creator) : MovableEntity(creator)
{
     RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
}
```

Framework

Beispielklasse: SUPER

- Damit nicht nur die Tick-Funktion von MyClass aufgerufen wird, sondern auch weiterhin der Tick von MovableEntity, müssen wir den Aufruf der Tick-Funktion an die Basisklasse weiterleiten:

```
CreateFactory(MyClass);

MyClass::MyClass(BaseObject* creator) : MovableEntity(creator)
{
     RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
     SUPER(MyClass, tick, dt);
}
```

Framework

Beispielklasse: orxout()

- Schlussendlich wollen wir noch etwas (sinnlose) Action in die Klasse bringen, daher geben wir in jedem Tick einen Text in die Konsole aus:

```
CreateFactory(MyClass);

MyClass::MyClass(BaseObject* creator) : MovableEntity(creator)
{
     RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

void MyClass::tick(float dt)
{
     SUPER(MyClass, tick, dt);

     orxout() << „Hello World“ << endl;
}
```

Framework

Ablauf dieser Einführung:

- Framework: Definition und Aufgaben
- Externe Libraries: Ogre, CEGui, Bullet
- Klassenhierarchie
- Worldentities
- Grafische Komponenten
- Beispielklasse

⇨ • XML

Framework

XML

- XML ist eine textbasierte Sprache, die die Interaktion zwischen verschiedenen Programmen und menschlichen Autoren ermöglicht.
- XML ermöglicht die Beschreibung von Attributen und Objekten.
- XML weist die selbe Form wie HTML auf.
- Wir verwenden XML, um Levels und andere Ansammlungen von Klassen (z.B. HUDs) zu beschreiben.
- Beispiel:

```
<MyClass myvalue="1" myothervalue="Hello World">
     <subclasses>
          <OtherClass somevalue="1.111" />
          <OtherClass somevalue="2.222" />
     </subclasses>
</MyClass>
```

Framework

XMLPort

- XMLPort ist unser Interface zwischen XML und C++.
- In XMLPort wird definiert, welche Objekte und Attribute in XML beschrieben werden können. Ausserdem werden Funktionen definiert, um diese Attribute lesen und schreiben zu können.
- Für jeden Wert braucht es ein Paar von set- und get-Funktionen. Die set-Funktion setzt den Wert im Objekt, die get-Funktion liest ihn aus.

- Beispiel:

```
void MyClass::XMLPort(...)
{
     SUPER(MyClass, XMLPort, ...);

     XMLPortParam(MyClass, "myvalue", setValue, getValue, xmlelement, mode);
     XMLPortParam(MyClass, "myothervaluevalue", setOtherValue, get...

     XMLPortObject(MyClass, OtherClass, "subclasses", addSubclass, getSubclass, xmlelement, mode);
}
```