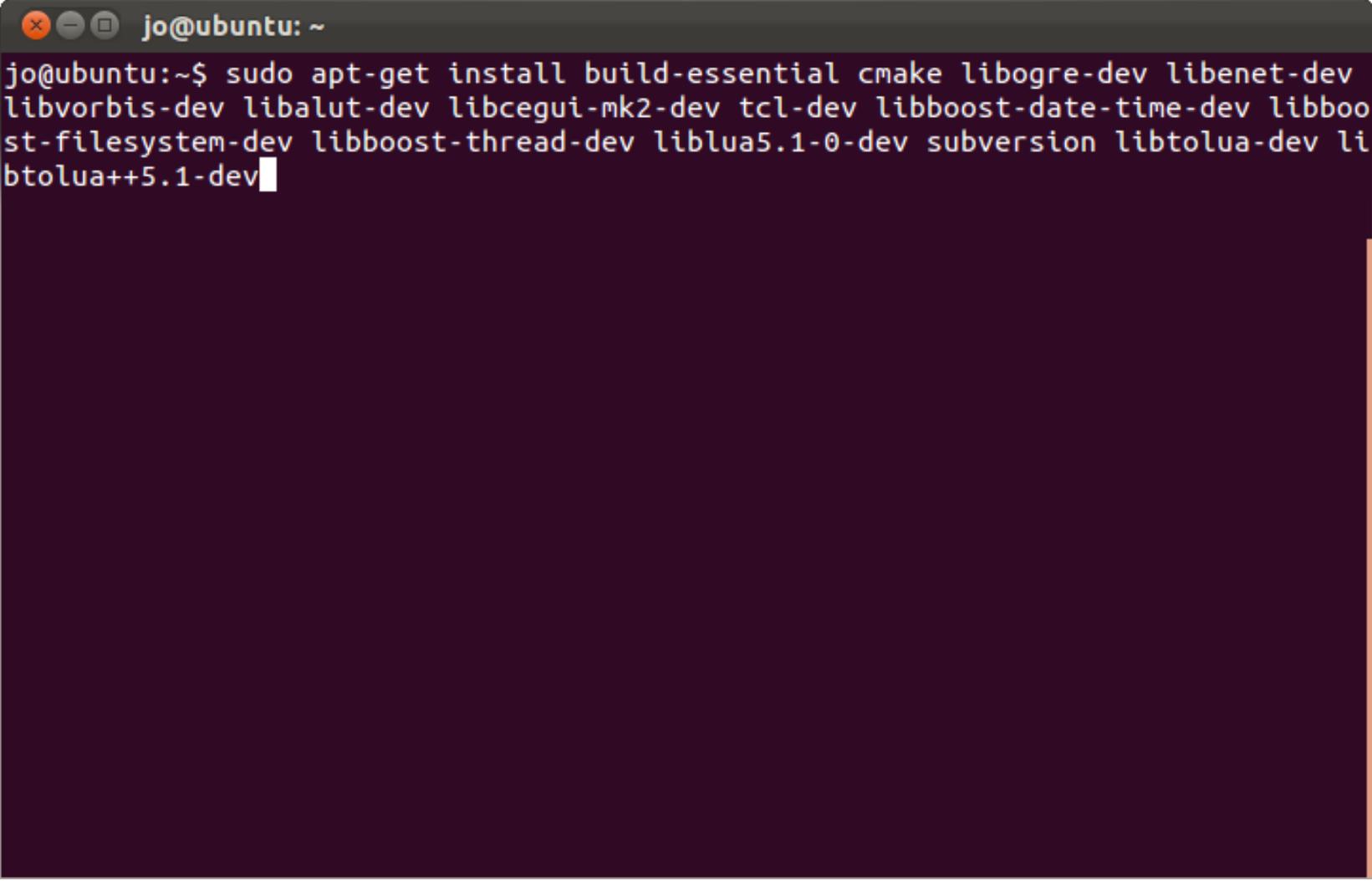


Framework & Coding

Einleitung in das Framework von Orxonox

Installation – Libraries



```
jo@ubuntu: ~  
jo@ubuntu:~$ sudo apt-get install build-essential cmake libogre-dev libenet-dev  
libvorbis-dev libalut-dev libcegui-mk2-dev tcl-dev libboost-date-time-dev libboo  
st-filesystem-dev libboost-thread-dev liblua5.1-0-dev subversion libtolua-dev li  
btolua++5.1-dev
```

Libraries

Ogre (Grafikengine)



Libraries

Ogre (Grafikengine)



Libraries

Ogre (Grafikengine)



Libraries

Ogre (Grafikengine)



Libraries

CEGui (GUI-Engine)

The screenshot displays a CEGui control panel titled "Demo 6 - Control Panel" with a close button (X) in the top right corner. The panel is divided into several sections:

- Table:** A table with three columns: "Column 1", "Column 2", and "Test. 2". The first row contains "Test Data", "1234567890", and "Third Column Entry". The second row contains "Test item", "Another item", and "More items". The third row contains "Abcdefg", "More items", and an empty cell. A mouse cursor is positioned over the "Test. 2" header.
- Column Control:** Includes fields for "ID Code:", "Width:", and "Caption:", each followed by an input box and an "Add" button. Below these is an "ID Code:" field with a "Delete Column" button.
- Row Control:** Includes fields for "Col ID:" and "Item Text:", each followed by an input box and an "Add" button. Below these is a "Row Idx:" field with a "Delete Row" button.
- Item Modification:** Includes fields for "Row Idx:", "Col ID:", and "Item Text:", each followed by an input box and a "Set" button.
- Statistics:** A green box at the bottom left displays performance metrics: "Current FPS: 263.736", "Average FPS: 263.195", "Worst FPS: 47.4543 1616 ms", "Best FPS: 277.168 3 ms", and "Triangle Count: 1254".
- Buttons:** A "Quit This Demo!" button is located at the bottom center of the panel.

The background of the control panel features a dark, textured pattern. The OGRE logo is visible in the bottom right corner.

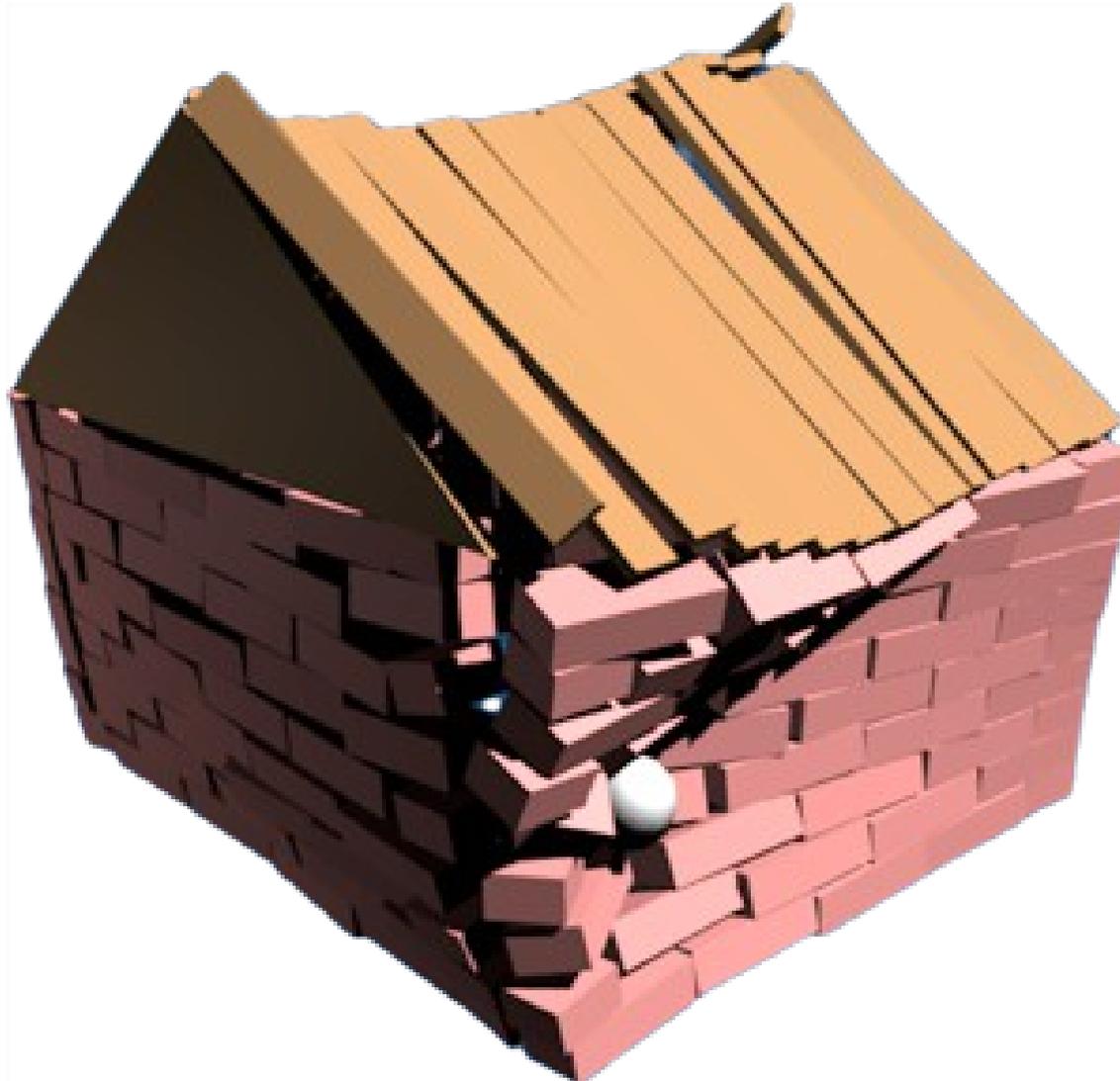
Libraries

CEGui (GUI-Engine)



Libraries

Bullet (Physikengine)



Objekteigenschaften - Raum

- jedes Objekt, das im Level einen Ort hat, erbt von der Basisklasse „Worldentity“.
 - jedes Objekt, das man sehen kann
- Worldentity: Punkt und Vektor im Raum
- Worldentities:
 - Statisch: StaticEntity (z.B. eine Spacestation)
 - Beweglich: MovableEntity (z.B. ein Projektil)
 - Kontrolliert: ControllableEntity (z.B. ein Spaceship)

Framework

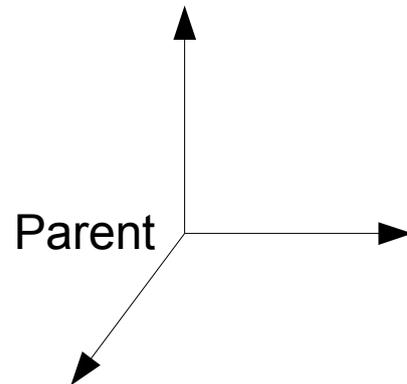
Worldentities: Definition

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

Framework

Worldentities: Attachen

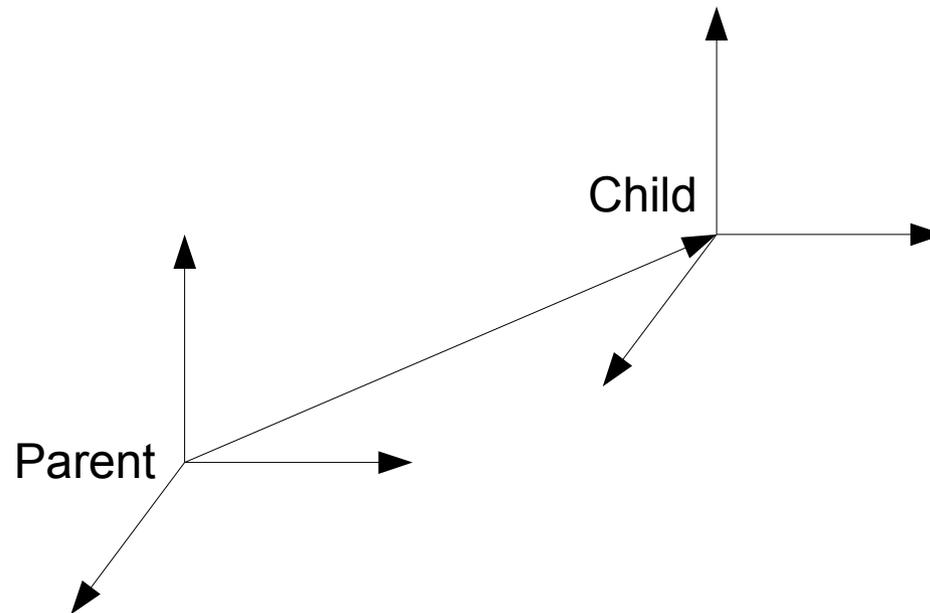
- Absolute Position im Raum (Parent):



Framework

Worldentities: Attachen

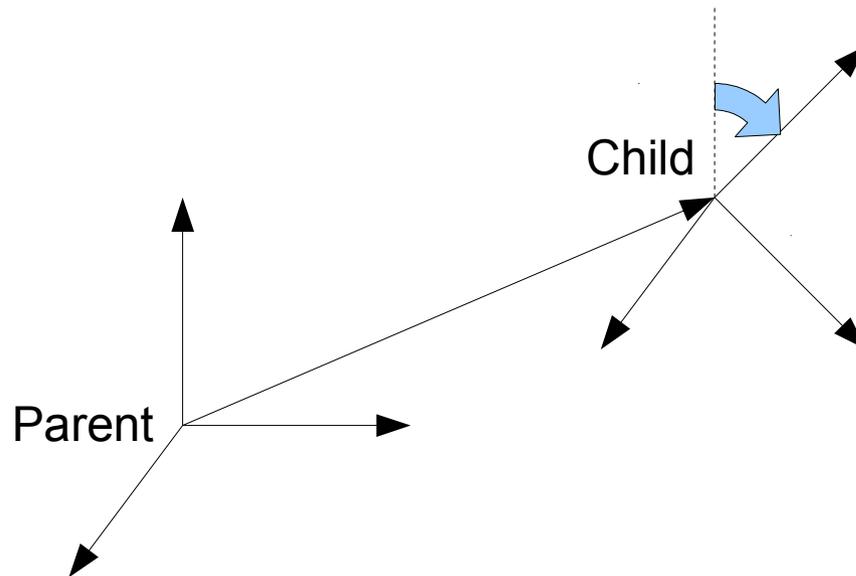
- Relative Position im Raum (Child):



Framework

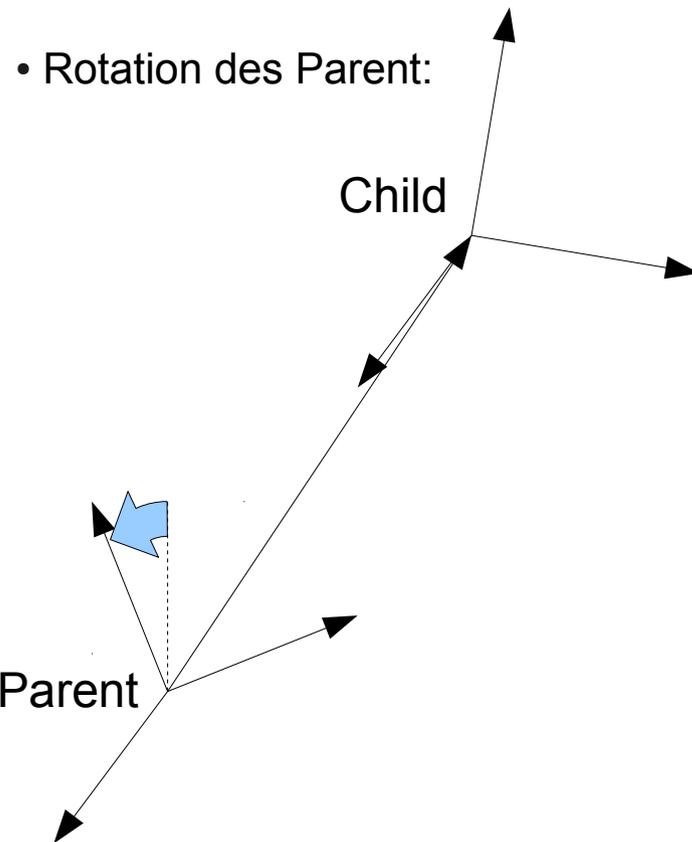
Worldentities: Attachen

- Rotation des Child:



Framework

Worldentities: Attachen



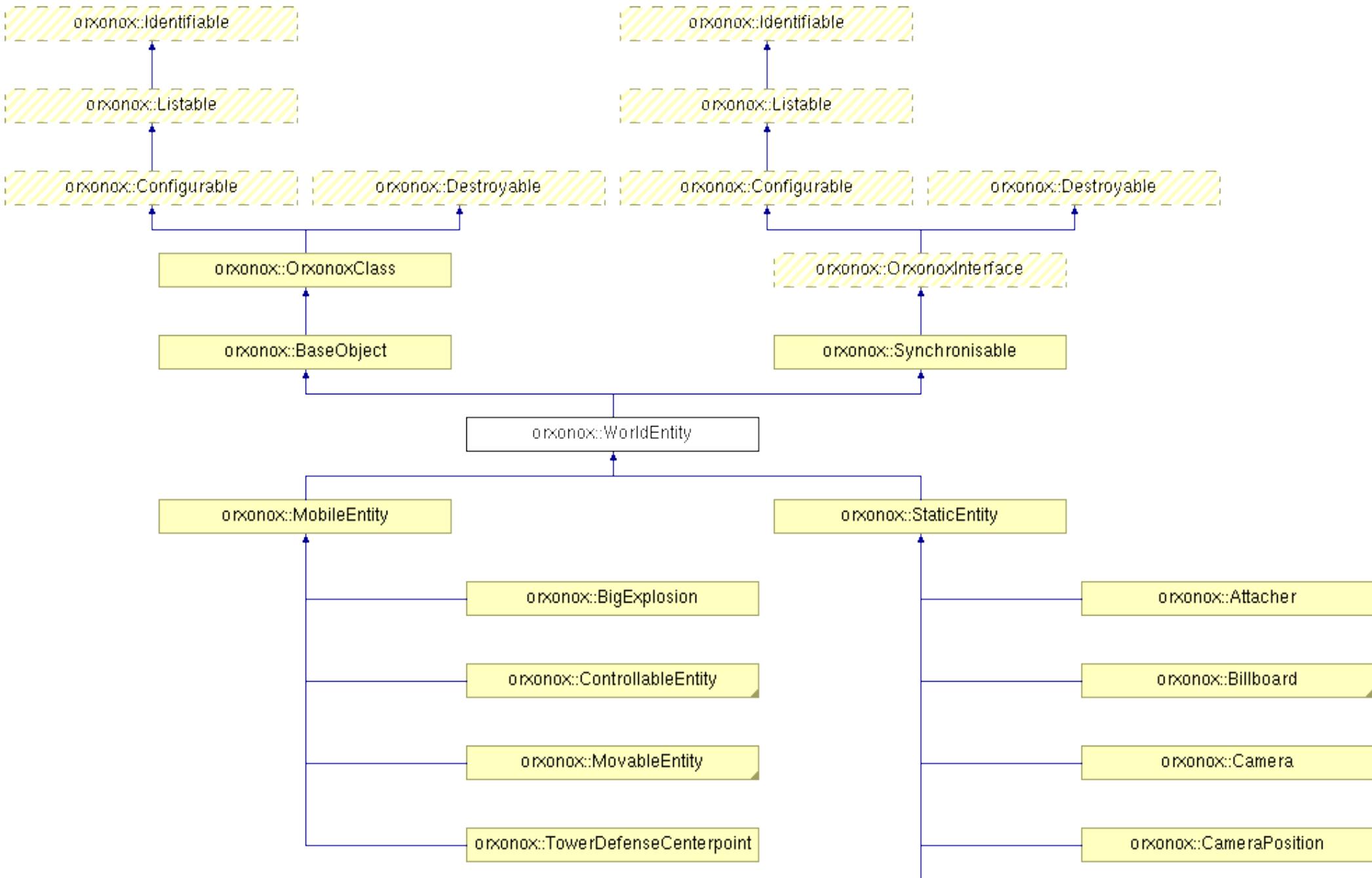
Objekteigenschaften - Zeit

- ein Objekt ändert sich in der Zeit -> es muss vom Interface Tickable erben.
- Klassen die von Tickable erben, erben die Funktion tick(float dt).
- ein Frame wird gerendert -> tick(dt) wird aufgerufen
- dt: die Zeit seit dem letzten Aufruf von tick(dt)

Objektmodellierung

- is–a–Relation:
 - ein Spaceship ist ein Worldentity
 - Mechanismus: Vererbung
- has–a–Relation:
 - „ein Spaceship hat Waffen, einen Antrieb, ...“
 - Mechanismus: Pointer auf ein anderes Objekt

Klassenhierarchie



Klassenhierarchie - OrxonoxClass

- alle Klassen und Interfaces erben von OrxonoxClass
- notwendig für das Funktionieren des Frameworks -> Servicefunktionalität

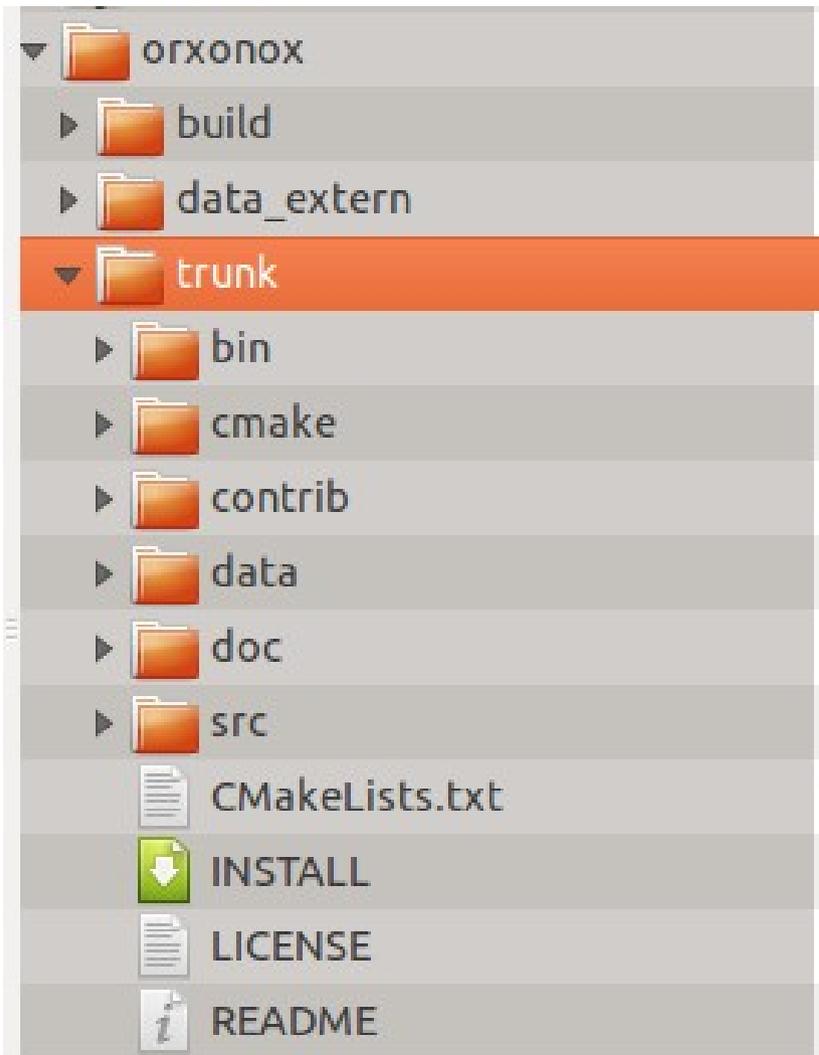
Klassenhierarchie - Baseobject

- Basisklasse aller „Objects“ in Orxonox
- Objects:
 - Können in ein Level geladen werden.
 - Werden am Ende des Levels wieder gelöscht.

CMake

- Findet die benötigten Libraries
- Erstellt ein Makefile
- Kann IDE-Projekt-Dateien erstellen

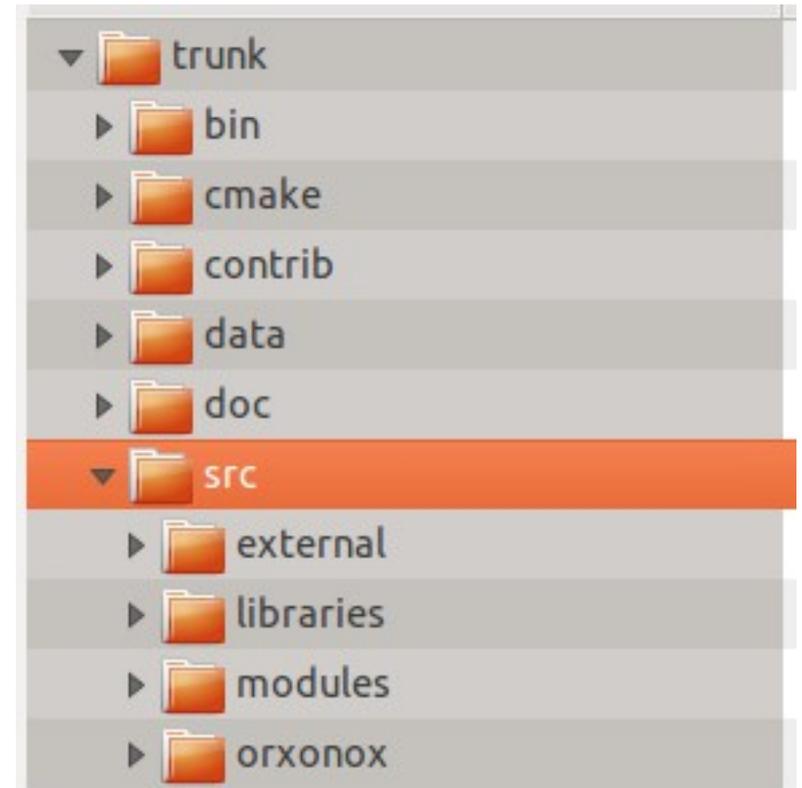
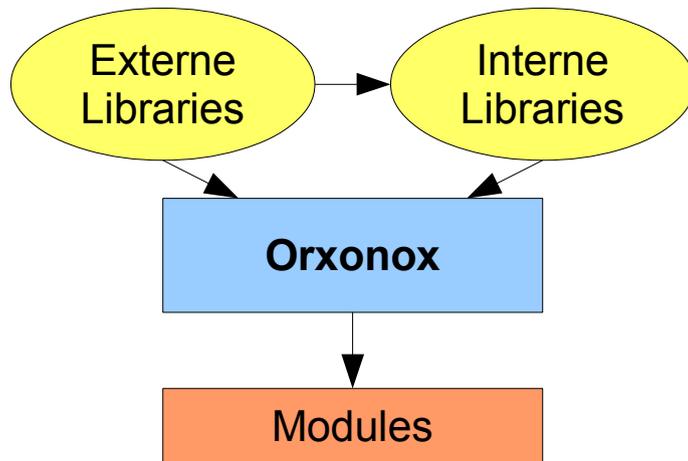
Verzeichnisstruktur - Trunk



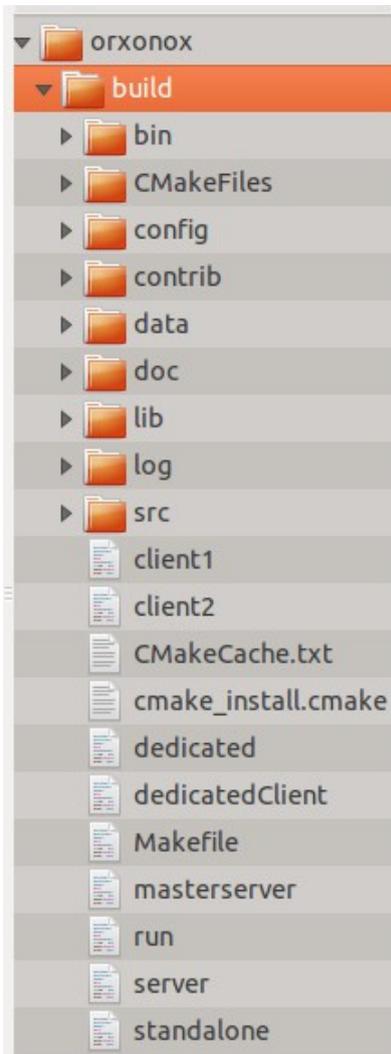
- cmake: CMake Scripts
- data: XML und Lua Scripts
- src: Quellcode

Framework

Struktur



Verzeichnisstruktur - Build



- bin: Executables
- config: Config-Files
- log: Output
- run: Startet Orxonox (runscript)

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(Context* context);
        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(Context* context)
{
}

MyClass::~MyClass()
{
}

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

Framework

Beispielklasse: RegisterClass

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

```
RegisterClass(MyClass) ;
```

```
MyClass::MyClass(Context* context)  
{  
}
```

```
MyClass::~MyClass()  
{  
}
```

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

Framework

Beispielklasse: RegisterObject

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

```
RegisterClass(MyClass);

MyClass::MyClass(Context* context)
{
    RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

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

Framework

Beispielklasse: Context

Ausserdem müssen wir den context-Pointer an die Basisklasse weitergeben:

```
RegisterClass(MyClass);

MyClass::MyClass(Context* context) : MovableEntity(context)
{
    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:

```
RegisterClass(MyClass);

MyClass::MyClass(Context* context) : MovableEntity(context)
{
    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:

```
RegisterClass(MyClass);

MyClass::MyClass(Context* context) : MovableEntity(context)
{
    RegisterObject(MyClass);
}

MyClass::~MyClass()
{
}

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

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

Framework

XML

- XML ist eine textbasierte Sprache, um Objekte zu speichern und zu laden.
- 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);
}
```