
Teil 4
© 2012-17 by Thomas Helml

Inhaltsverzeichnis

Ziele

- Links
  - libGDX: http://libgdx.badlogicgames.com
  - Wiki: https://github.com/libgdx/libgdx/wiki
- libGDX ist
  - ein Java Game Development Framework
  - eine API für alle Plattformen
  - keine Spielengine (wie z.B. Unity)

Features

- Cross Plattform
  - Windows
  - Linux
  - Mac OS X
  - Android (2.2+)
  - BlackBerry
  - iOS
  - Java Applet
  - JavaScript/WegGL

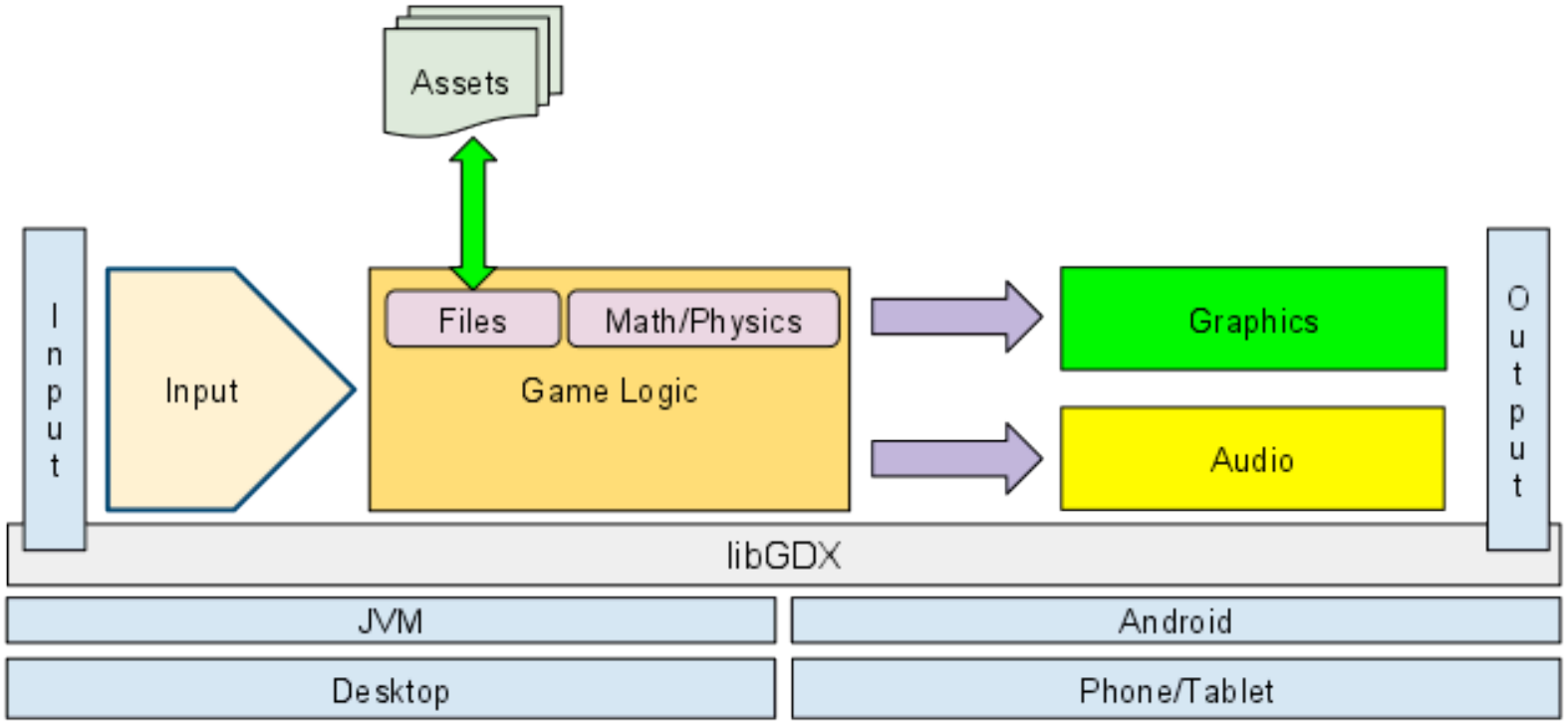
Features

- Audio
  - Streaming + Sound FX (ogg, wav, mp3)
  - direkter HW-Zugriff für PCM Audio
- Input
  - Abstraktion für Maus, Touch, Keyboard, Gyro, Kompass
  - Gesture Detection

Features

- Math+Physik Engine
- File I/O
  - Abstrahiertes File System für alle Plattformen
- Grafik
  - OpenGL ES 2.0 (OpenGL for Embedded Systems)
  - 2D API (z.B. TMX Map Support, ...)
  - 3D API

Das Framework



Das Framework

- 6 wichtige Module:
  - Application
  - Files
  - Input
  - Net
  - Audio
  - Graphics

Das Framework

- Application
  - Startet Anwendung
  - Informiert über Events auf App-Ebene
    - z.B. Fenstergrößenänderung
  - Logging
  - Abfrage Methoden
    - z.B. Speichergröße

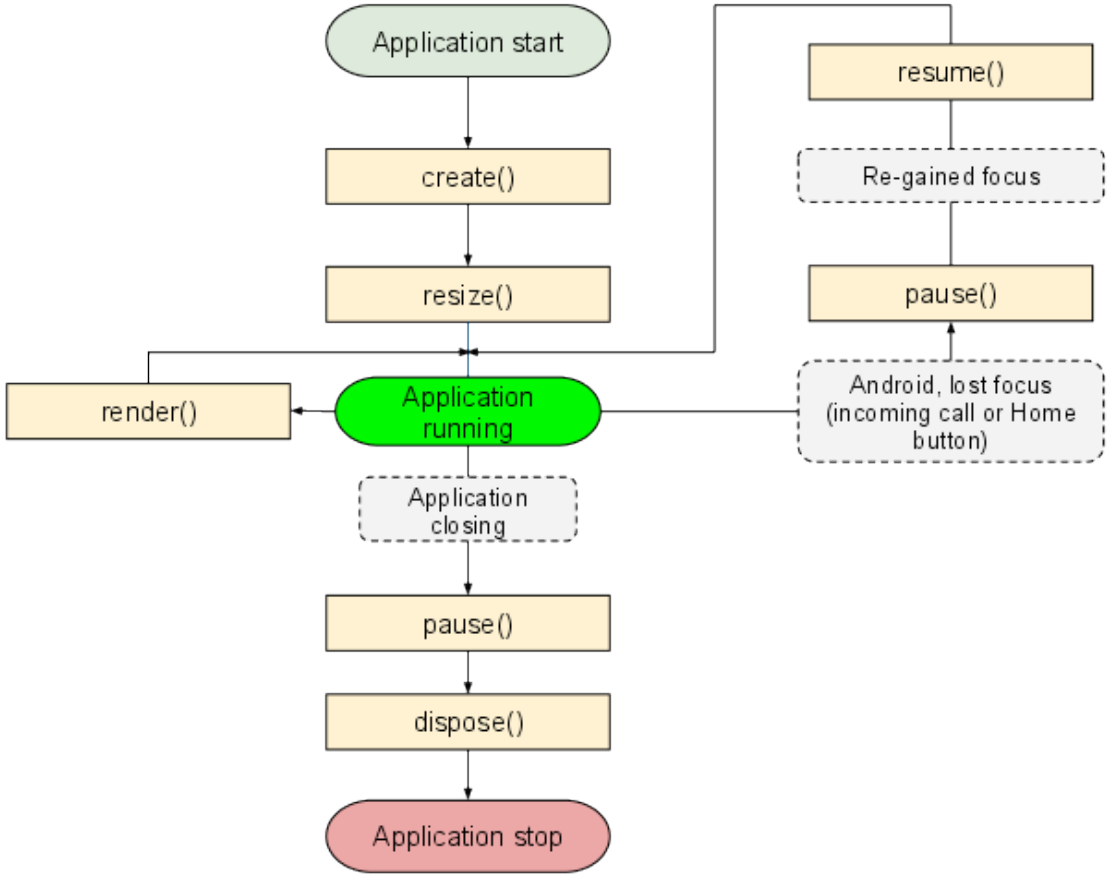
Das Framework

- Files
  - Abstrahiert Dateizugriff
  - Methoden inkompatibel mit Java File Klassen
  - Bsp:
- Input
  - Maus, Tastatur, Touch, Gyroskop, ...
  - Polling und Ereignissteuerung

Das Framework

- Net
  - Zugriff auf Ressourcen über HTTP(S)
  - Socket Kommunikation
- Audio
  - Abspielen von Sound FX und Musik
  - Direkter HW Zugriff über PCM Audio
- Graphics
  - Zugriff auf OpenGL ES 2.0
  - Video Modus ändern, ...

libGDX-Life-Cycle



libGDX-Life-Cycle

- jedes Spiel muss das Interface ApplicationListener (bzw. ApplicationAdapter) implementieren
- diese Klasse muss in den sog. Starterklassen registriert werden
- die App ruft je nach Ereignis die entsprechenden Methoden auf

libGDX-Life-Cycle

```
public class MyGame implements ApplicationListener {
    public void create () {}
    public void render () {}
    public void resize (int width, int height) {}
    public void pause () {}
    public void resume () {}
    public void dispose () {}
}
```

libGDX-Life-Cycle

- create ()
  - Aufruf nach Anwendungsstart
- resize(int width, int height)
  - Aufruf nach create()
  - jedes Mal, wenn Fenstergröße geändert wird
- dispose()
  - Aufruf, wenn Anwendung beendet wird

libGDX-Life-Cycle

- render ()
  - Aufruf in unregelmäßigen Abständen
  - immer wenn Spiel neu gezeichnet werden muss
  - Spiellogik muss hier behandelt werden
  - Entspricht Konzept der „Gameloop“

libGDX-Life-Cycle

- pause()
  - Wichtig für Android
  - Aufruf wenn App in Hintergrund geht (Telefon, Homebutton, ...)
  - Spielstatus muss/soll hier gesichert werden
  - Am Desktop: Aufruf nur vor dispose()
- resume()
  - Nur Android: Aufruf nach Ende von pause()

Introduction to libGDX

- http://youtu.be/BTC922ki2mc

Aufgabe

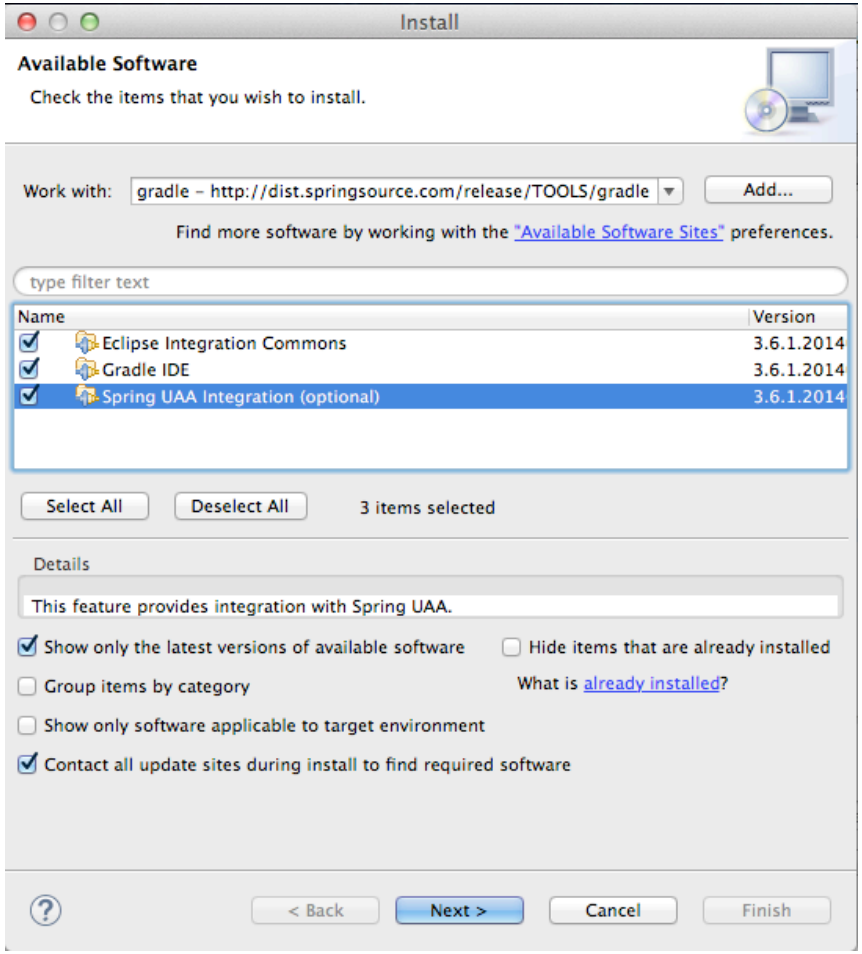
- Download libGDX
  - http://libgdx.badlogicgames.com/download.html
- Projekt Test_libGDX erzeugen+starten

Projekterstellung

- `gdx-setup.jar`
- Plattform auswählen
- aus Zielordner wird Projekt importiert
- Gradle ist notwendig
  - Paketabhängigkeit

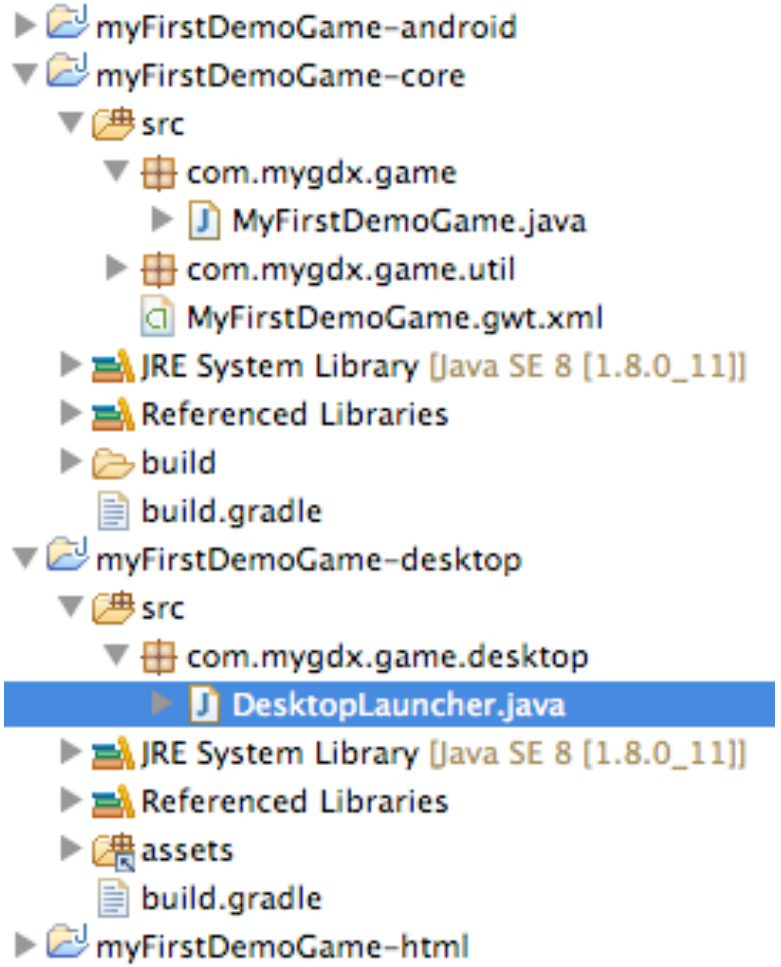
Projekterstellung

- Eclipse: Help – Install new Software
- Nach Installation:
  - File – Import – Gradle – Gradle Import
  - Erzeugtes Projekt auswählen



Starter Classes

- einziger Code, der plattformabhängig ist
- jeweils eigenes Projekt in Eclipse



Starter Classes

- Beispiel Desktop

```
public class DesktopStarter {
    public static void main(String[] argv)
    {
        LwjglApplicationConfiguration config =
            new LwjglApplicationConfiguration();
        new LwjglApplication(new MyGame(), config);
    }
}
```

Starter Classes

- Beispiel Android

```
public class AndroidStarter extends AndroidApplication{
    public void onCreate(Bundle bundle){
        super.onCreate(bundle);
        AndroidApplicationConfiguration config =
            new AndroidApplicationConfiguration();
        initialize(new MyGame(), config);
    }
}
```

create()

```
public void create () {
    batch = new SpriteBatch();
    img = new Texture("badlogic.jpg");
}
```

- `SpriteBatch`:
  - Benötigt um 2D Rechtecke zu zeichnen (Textures)
  - Alle Zeichenoperationen eines SpriteBatch beziehen sich auf Screenkoordinaten
  - Links unten Null-Punkt
  - max. ein SpriteBatch Objekt je Spiel
  - muss freigegeben werden (dispose)

create()

```
public void create () {
    batch = new SpriteBatch();
    img = new Texture("badlogic.jpg");
}
```

- `Texture`:
  - Grafik, die in den Grafikspeicher geladen wird
  - Grafiken MÜSSEN Breite/Höhe haben, die 2er Potenzen sind (16x16, 64x256 ...)

render()

```
public void render () {
    Gdx.gl.glClearColor(1, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    batch.begin();
    batch.draw(img, 0, 0);
    batch.end();
}
```

- `glClearColor(1,0,0,1);`
  - RGBA(1,0,0,1) => Rot
  - letzter Parameter für Alpha-Kanal
- `Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);`
  - OpenGL Befehl für Bildschirm löschen

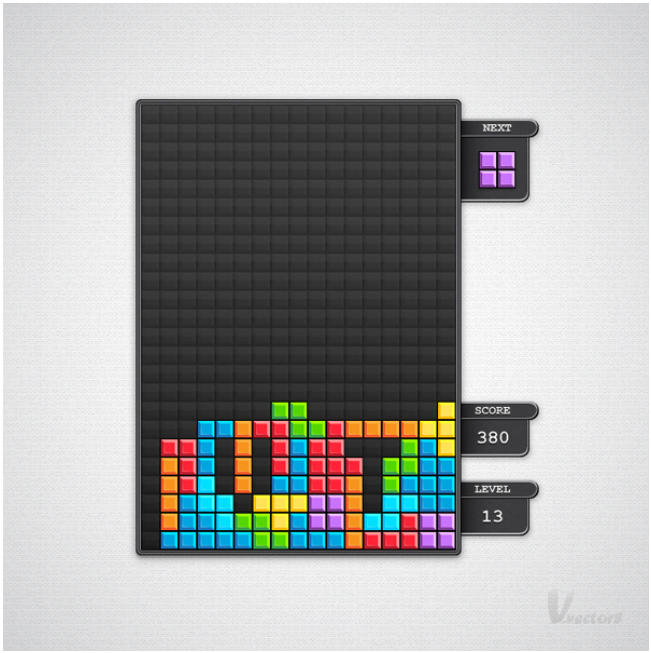
render()

```
public void render () {
    Gdx.gl.glClearColor(1, 0, 0, 1);
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
    batch.begin();
    batch.draw(img, 0, 0);
    batch.end();
}
```

- `begin(), end();`
  - alle Zeichenoperationen müssen zw. `begin()` und `end()` stehen
- `draw(img, 0,0);`
  - zeichnet Texture an der Position (0/0)

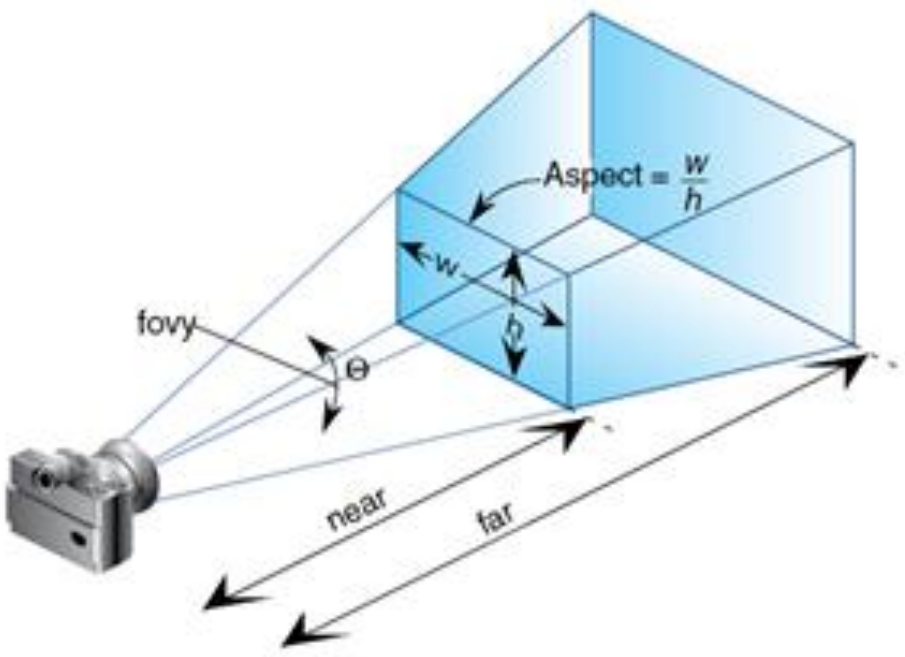
Grafiken

- http://design.tutsplus.com/tutorials/block-game-interface--vector-5269



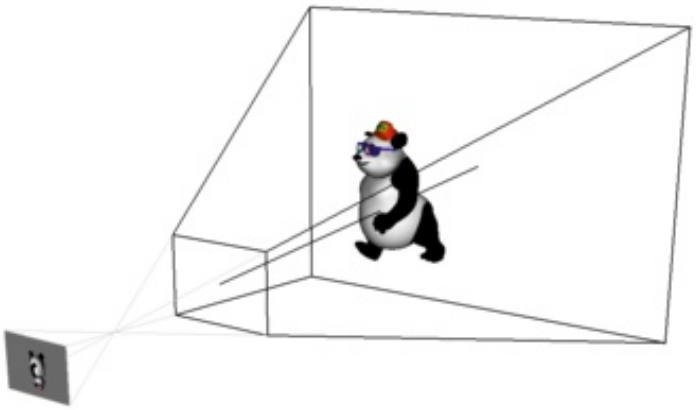
OrthographicCamera

- Camera „sieht“ Ausschnitt aus realer Welt

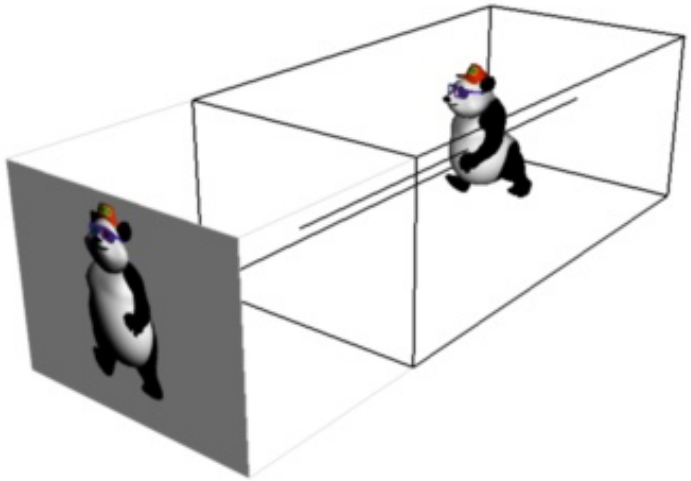


OrtographicCamera

Perspektivische Projektion



Orthografische Projektion

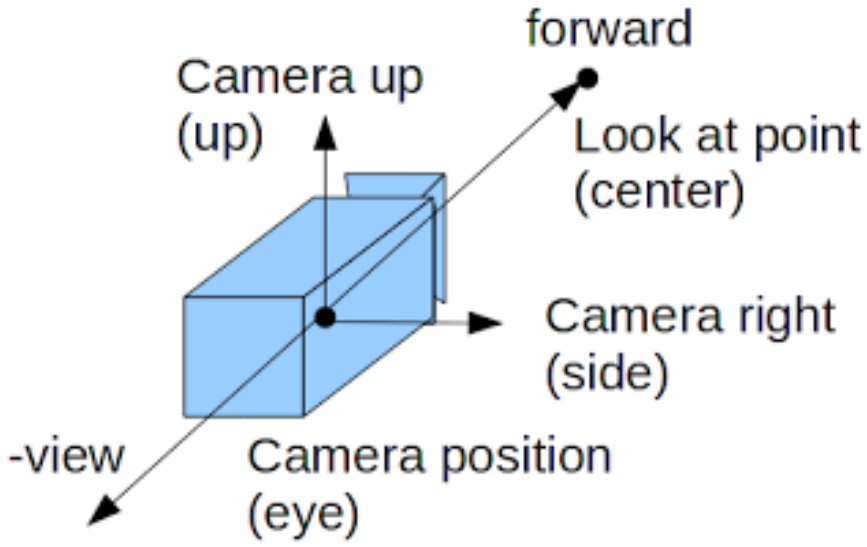


OrtographicCamera

- Camera ist vergleichbar mit Kamera IRL
  - Bewegen
  - Rotieren
  - Zoom in/out
  - Ändern des Betrachtungspunktes
  - ...

OrthographicCamera

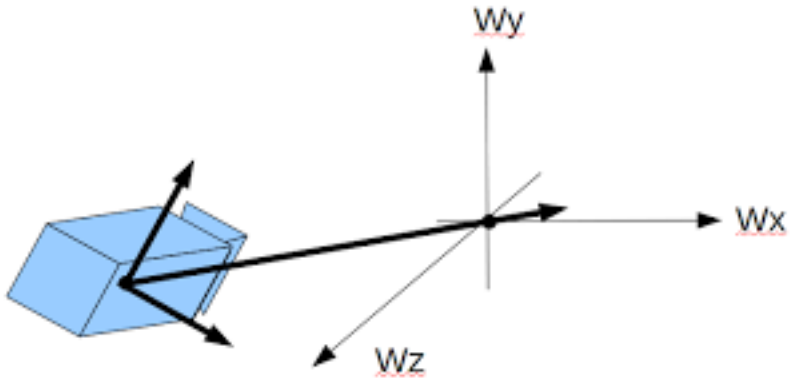
- Position der Camera
  - Raumkoordinaten (x,y,z)




```
cam = new OrthographicCamera( ... );
cam.position.set(float x, float y, float z);
cam.update();
```

OrthographicCamera

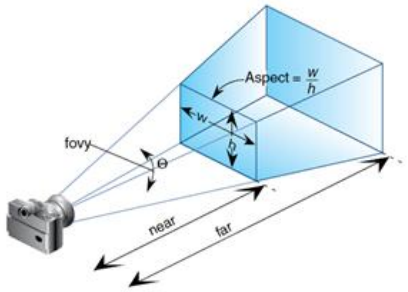
- Camera hat Ausrichtung
  - dh. sie „schaut“ auf bestimmte Koordinate in der Welt




```
cam = new OrthographicCamera( ... );
cam.lookAt(float x, float y, float z);
cam.update();
```

OrthographicCamera

- Camera kann im Raum bewegt werden




```
cam = new OrthographicCamera( ... );
cam.translate(float x, float y, float z);
cam.update();
```

OrthographicCamera

- Rotieren der Camera:

```
cam.rotate (  float angle,
              float axisX,
              float axisY,
              float axixZ);
```

- Rotation um Achse (bestimmt durch Vektor) um `angle` Grad

OrthographicCamera

- Render-Methode

```
@Override
 public void render() {
     handleInput();
     // camera Position, etc. aktualisieren
     cam.update();
     // „Zeichenfläche“ aktualisieren mit neuer
     // Camera-Projektion
     batch.setProjectionMatrix(cam.combined);

     Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);

     batch.begin();
     mapSprite.draw(batch);
     batch.end();
 }
```

OrthographicCamera

Constructor:

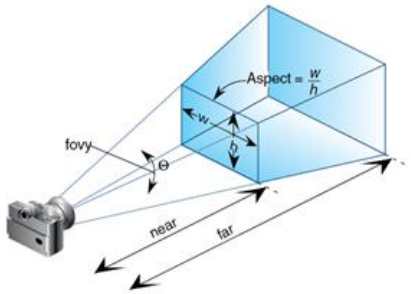
```
public OrthographicCamera(float viewportWidth,
                          float viewportHeight)
```

Parameters:

viewportWidth - viewport Breite
viewportHeight - viewport Höhe

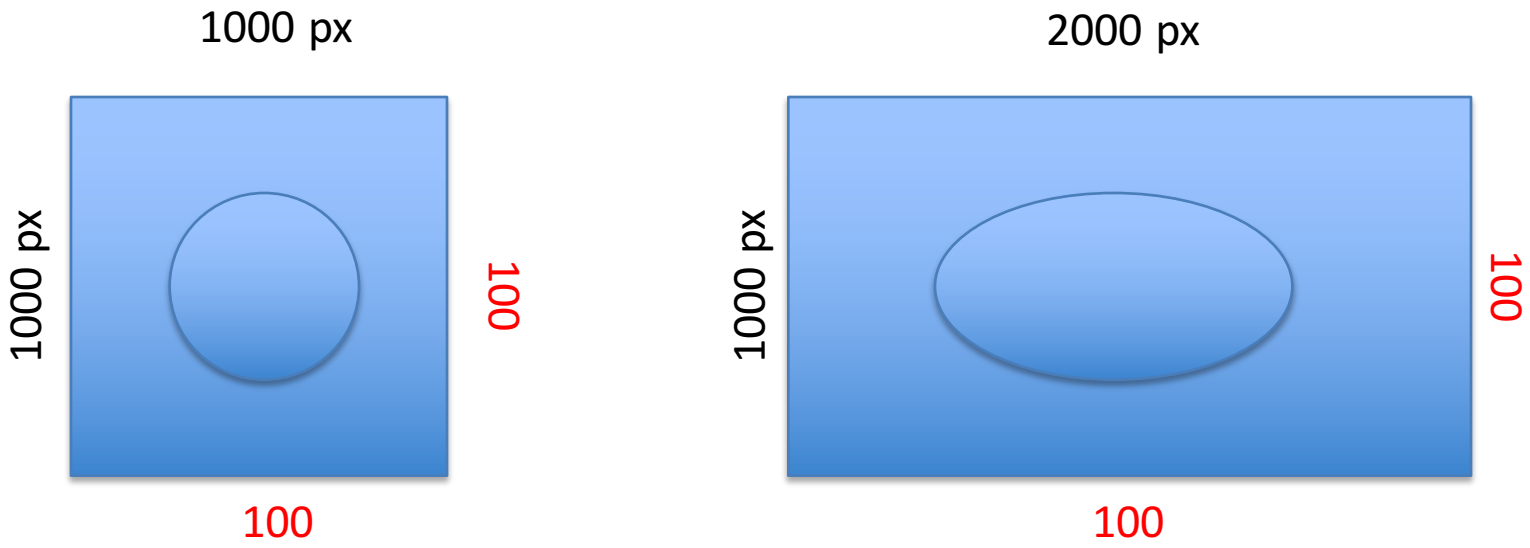
OrthographicCamera

- viewPort:
  - Größe des sichtbaren Bereichs (=Fenster)
  - Angabe in „Welteinheiten“ (was sieht man)
    - Einheit z.B. m, cm, Bausteinbreite, ...
    - nicht in Pixel denken!
  - Beispiel: Fullscreen
    - Tatsächl. Auflösung: 800x600 Pixel
      - Weltkoordinaten – sichtbarer Bereich: 40x30m
    - Was passiert, wenn Auflösung geändert wird?



OrthographicCamera

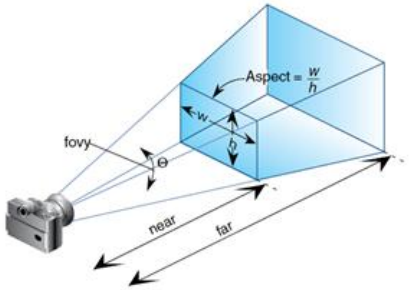
- Bildschirm:



OrthographicCamera

- LÖSUNG
  - Viewport muss Proportionen des Fensters berücksichtigen!
    - Breite/Höhe, z.B.: 4:3
  - Breite/Höhe des Fensters:

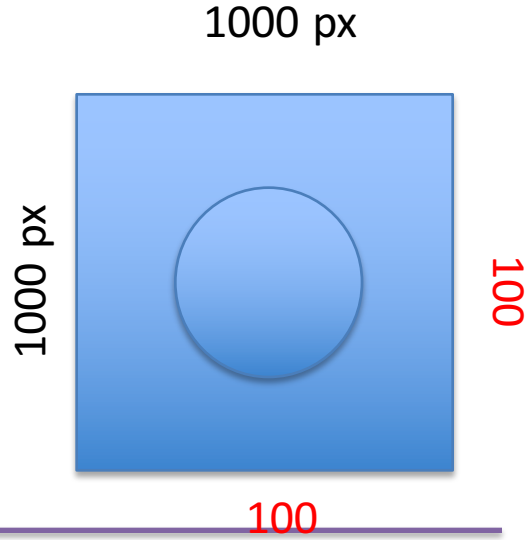
```
Gdx.graphics.getWidth();
Gdx.graphics.getHeight();
```



OrthographicCamera

```
cam = new OrthographicCamera (
            100, 100 * (Gdx.graphics.getHeight() /
                 Gdx.graphics.getWidth()));
```

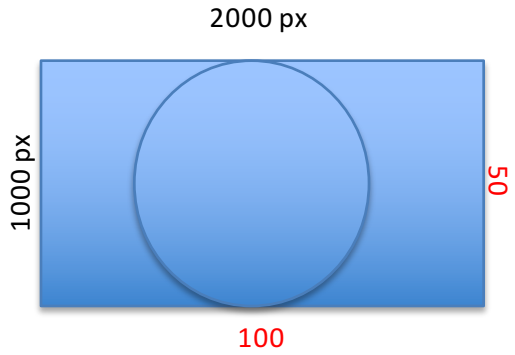
- Beispiel:
  - Auflösung: Breite = 1000px, Höhe= 1000px
  - Höhe/Breite = 1000/1000 = 1.0
  - viewportWidth = 100
  - viewPortHeight = 100*1.0 = 100



OrthographicCamera

```
cam = new OrthographicCamera (
            100, 100 * (Gdx.graphics.getHeight() /
                 Gdx.graphics.getWidth()));
```

- Beispiel:
  - Auflösung: Breite = 2000px, Höhe= 1000px
  - Höhe/Breite = 1000/2000 = 0.5
  - viewportWidth = 100
  - viewPortHeight = 100 * 0.5 = 50



Sprite

- Sprite vs. Texture
  - Sprite braucht eine Texture (=Grafik)
  - Sprites können vergrößert/-kleinert werden
  - Sprites können rotiert werden
  - Sprites können transparent werden
- Texture = Oberfläche
- Sprite = Stuktur

Sprite

- Klasse `Sprite`
  - beinhaltet
    - Farbe
    - Texture
    - Geometrie (Rotieren, Skalieren, ...)
  - in einem Objekt
  - Position muss mit `.setPosition()` gesetzt werden!

Sprite

- Größe der Texturen in der Welt bestimmen:

```
Sprite.setSize(WORLD_WIDTH,WORLD_HEIGHT);
```

- Create-Methode:

```
mapSprite = new Sprite(new Texture(
                    Gdx.files.internal("sc_map.png")));
mapSprite.setPosition(0, 0);
mapSprite.setSize(WORLD_WIDTH, WORLD_HEIGHT);
```

Networking

- Kommunikation über Sockets
- Interface Socket:

isConnected

```
boolean isConnected()
```

Returns:

whether the socket is connected

getInputStream

```
java.io.InputStream getInputStream()
```

Returns:

the InputStream used to read data from the other end of the connection.

getOutputStream

```
java.io.OutputStream getOutputStream()
```

Returns:

the OutputStream used to write data to the other end of the connection.

Networking

- ServerSocket

accept

`Socket accept(SocketHints hints)`

Accepts a new incoming connection from a client `Socket`. The given hints will be applied to the accepted socket. Blocking, call on a separate thread.

Parameters:

`hints` - additional `SocketHints` applied to the accepted `Socket`. Input null to use the default setting provided by the system.

Returns:

the accepted `Socket`

Throws:

`GdxRuntimeException` - in case an error occurred

Networking

- Sockets erzeugen:
  - Interface Net

newServerSocket

```
ServerSocket newServerSocket(Net.Protocol protocol,
                             int port,
                             ServerSocketHints hints)
```

Creates a new server socket on the given port, using the given `Net.Protocol`, waiting for incoming connections.

Parameters:

`port` - the port to listen on

`hints` - additional `ServerSocketHints` used to create the socket. Input null to use the default setting provided by the system.

Returns:

the `ServerSocket`

Throws:

`GdxRuntimeException` - in case the socket couldn't be opened

Networking

- Sockets erzeugen:
  - Interface Net

newClientSocket

```
Socket newClientSocket(Net.Protocol protocol,
                       java.lang.String host,
                       int port,
                       SocketHints hints)
```

Creates a new TCP client socket that connects to the given host and port.

Parameters:

host - the host address

port - the port

hints - additional SocketHints used to create the socket. Input null to use the default setting provided by the system.

Returns:

GdxRuntimeException in case the socket couldn't be opened

Networking

```
String textToSend = new String("TEST 123 ;)");

SocketHints socketHints = new SocketHints();
// Socket Time out  4s
socketHints.connectTimeout = 4000;
// Socket erzeugen und mit server connecten
Socket socket = Gdx.net.newClientSocket(
                    Protocol.TCP,
                    "localhost",
                    9024,
                    socketHints);
try {
    // Nachricht schicken
    socket.getOutputStream().write(textToSend.getBytes());
} catch (IOException e) {
        e.printStackTrace();
}
```

Networking

```
ServerSocketHints serverSocketHint = new ServerSocketHints();
// 0 => kein timeout
serverSocketHint.acceptTimeout = 0;

// server socket erzeugen
ServerSocket serverSocket = Gdx.net.newServerSocket(Protocol.TCP, 9024,
serverSocketHint);

while(true){
    // socket für Kommunikation
    Socket socket = serverSocket.accept(null);

    // daten in BufferedReader lesen
    BufferedReader buffer = new BufferedReader (
                                new
InputStreamReader(socket.getInputStream()));
    try {
        System.out.println(buffer.readLine());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

JSON

- JavaScript Object Notation
  - Datenformat zum Datenaustausch in Textform
  - konzipiert für JavaScript

```
public class Person {
    private String name;
    private int age;
    private ArrayList numbers;
}
public class PhoneNumber {
    private String name;
    private String number;
}
```

```
Person person = new Person();

person.setName("Nate");
person.setAge(31);
ArrayList numbers = new ArrayList();
numbers.add (new PhoneNumber(
        "Home", "206-555-1234"));
numbers.add (new PhoneNumber(
        "Work", "425-555-4321"));
person.setNumbers(numbers);
```

JSON

- Repräsentation als JSON Objekt:

```
public class Person {
    private String name;
    private int age;
    private ArrayList numbers;
}
public class PhoneNumber {
    private String name;
    private String number;
}
```

```
{
numbers: [
    {
        class: com.example.PhoneNumber,
        number: "206-555-1234",
        name: Home
    },
    {
        class: com.example.PhoneNumber,
        number: "425-555-4321",
        name: Work
    }
],
name: Nate,
age: 31
}
```

JSON

- JSON Objekt aus Java Objekt:

```
Json json = new Json();
String text = json.toJson(person));
```

- Java Objekt aus JSON Objekt:

```
Person p2 = json.fromJson(Person.class,
                          text);
```