

2D Animation

[Edit](#)[New Page](#)[Jump to bottom](#)

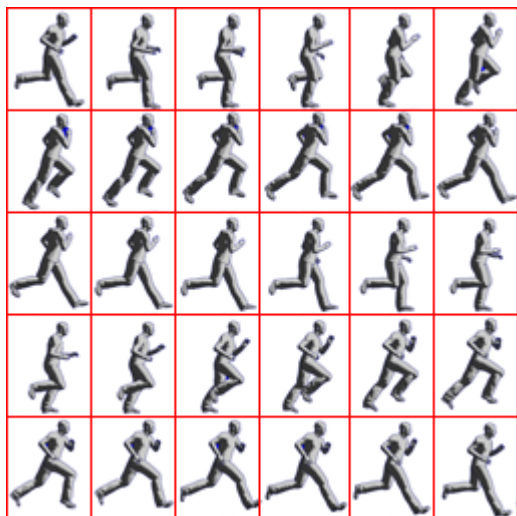
FlorentMasson edited this page on Apr 24, 2017 · 15 revisions

2D Animation is a technique used to create the illusion of movement using static images. This article describes how to create animations with libGdx using its [Animation Class](#).

Background

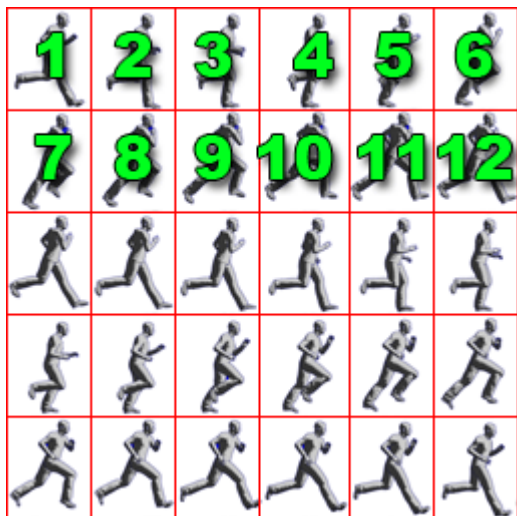
An animation consists of multiple frames which are shown in a sequence at set intervals. An animation of a running man can be achieved by taking pictures of him while running and playing those images back sequentially in a loop.

The following "sprite sheet" image shows a complete cycle of a man running. Each box contains a frame of animation. When these frames are shown sequentially over a period of time, they appear as an animated image.



The frame rate is how often the frame is changed per second. The example sprite sheet's complete running cycle has 30 frames (6 columns and 5 rows). If the character is to complete a cycle in one second, 30 frames must be shown per second, so the frame rate is 30 FPS. The time per frame (known as the frame time or interval time) is the reciprocal of the FPS, in this case 0.033 seconds per frame .

An animation is a very simple state machine. The running man has 30 states as per the sprite sheet. The numbered frames represent the states a running man goes through, only one at a time. The current state is determined by the amount of time since the animation began. If less than 0.033 seconds have elapsed, we are in State 1, so the first sprite is drawn. If we are between 0.033 and 0.067 seconds, then we are in State 2, and so on. If the animation is looping, it returns to the first frame after all frames have been shown.



The Animation class

LibGDX's [Animation \(code\)](#) class can be used to easily manage an animation. It is constructed with a list of images and the frame interval time. During playback, its `getKeyFrame` method takes an elapsed time parameter and returns the appropriate image for that time.

Animation has a generic type parameter for the type of class that represents the image. The type would typically be a `TextureRegion` or `PolygonRegion`, but any renderable object can be used. The type is declared by specifying the animation type in the Animation declaration, for example `Animation<TextureRegion> myAnimation = new Animation<TextureRegion>(/*...*/) .` Note that it would usually be inadvisable to use the `Sprite` class to represent frames of an animation, because the `Sprite` class contains positional data that would not carry from frame to frame.

TextureAtlas example

LibGDX's [TextureAtlas \(code\)](#) class is typically used for combining many separate `TextureRegions` into a smaller set of Textures to reduce expensive draw calls. ([details here](#)).

TexturePacker and TextureAtlas provide a convenient way to generate animations. All the source images of an animation should be named with an underscore and frame number at the end, such as `running_0.png`, `running_1.png`, `running_2.png`, etc. TexturePacker will automatically use these numbers as frame numbers (so long as the packing parameter `useIndexes` is left true).

After the TextureAtlas is loaded, a complete array of frames can be acquired at once and passed into the Animation constructor:

```
public Animation<TextureRegion> runningAnimation;

//...

runningAnimation =
    new Animation<TextureRegion>(0.033f, atlas.findRegions("running"), PlayMode.LOOP);
```

Sprite sheet example

The following code snippet will create an Animation using the `animation_sheet.png` sprite-sheet and renders the animation to the screen.

```
c class Animator implements ApplicationListener {

    // Constant rows and columns of the sprite sheet
    private static final int FRAME_COLS = 6, FRAME_ROWS = 5;

    // Objects used
    Animation<TextureRegion> walkAnimation; // Must declare frame type (TextureRegion)
    Texture walkSheet;
    SpriteBatch spriteBatch;

    // A variable for tracking elapsed time for the animation
    float stateTime;

    @Override
    public void create() {

        // Load the sprite sheet as a Texture
        walkSheet = new Texture(Gdx.files.internal("animation_sheet.png"));

        // Use the split utility method to create a 2D array of TextureRegions. This
        // possible because this sprite sheet contains frames of equal size and they
        // all aligned.
        TextureRegion[][] tmp = TextureRegion.split(walkSheet,
            walkSheet.getWidth() / FRAME_COLS,
```

```

        walkSheet.getHeight() / FRAME_ROWS);

    // Place the regions into a 1D array in the correct order, starting from the
    // left, going across first. The Animation constructor requires a 1D array.
    TextureRegion[] walkFrames = new TextureRegion[FRAME_COLS * FRAME_ROWS];
    int index = 0;
    for (int i = 0; i < FRAME_ROWS; i++) {
        for (int j = 0; j < FRAME_COLS; j++) {
            walkFrames[index++] = tmp[i][j];
        }
    }

    // Initialize the Animation with the frame interval and array of frames
    walkAnimation = new Animation<TextureRegion>(0.025f, walkFrames);

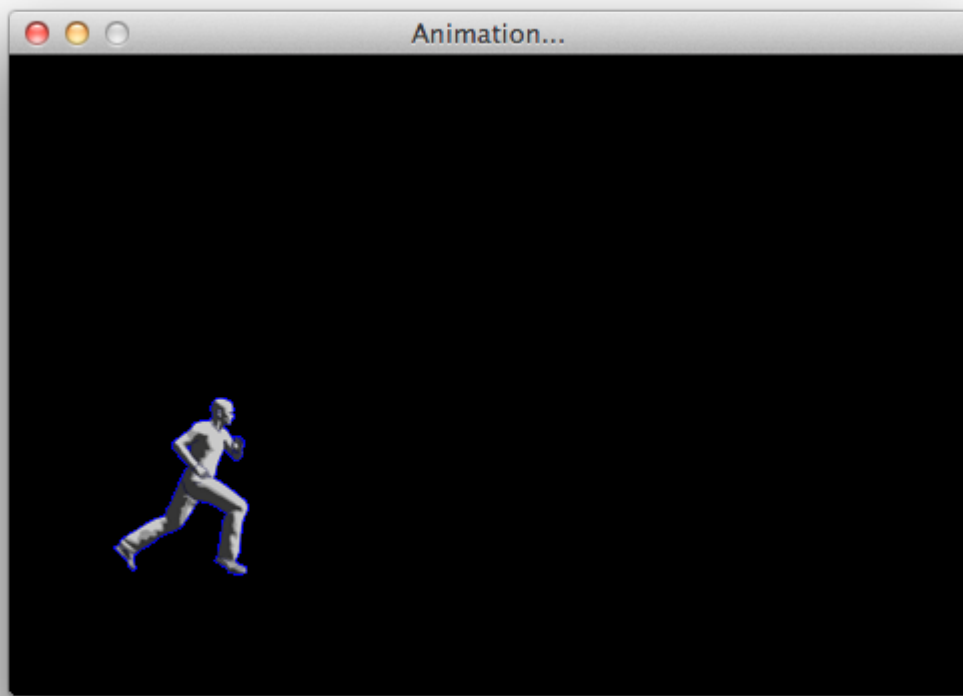
    // Instantiate a SpriteBatch for drawing and reset the elapsed animation
    // time to 0
    spriteBatch = new SpriteBatch();
    stateTime = 0f;
}

@Override
public void render() {
    Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT); // Clear screen
    stateTime += Gdx.graphics.getDeltaTime(); // Accumulate elapsed animation time

    // Get current frame of animation for the current stateTime
    TextureRegion currentFrame = walkAnimation.getKeyFrame(stateTime, true);
    spriteBatch.begin();
    spriteBatch.draw(currentFrame, 50, 50); // Draw current frame at (50, 50)
    spriteBatch.end();
}

@Override
public void dispose() { // SpriteBatches and Textures must always be disposed
    spriteBatch.dispose();
    walkSheet.dispose();
}

```



Creating an animation is extremely simple by using the following constructor.

Method signature	Description
<code>Animation (float frameDuration, TextureRegion... keyFrames)</code>	The first parameter is the frame time and the second is an array of regions (frames) making up the animation

Best practices

- Pack frames into one texture along with other sprites to optimize rendering. This is easily done with TexturePacker.
- Settle for a reasonable number of frames depending on the game type. For a retro arcade style, 10 fps may suffice, while more realistic looking movements require more frames.

Assets

Get the sprite-sheet [here](#).



► Pages 216

Table of Contents



- [Wiki Style Guide](#)
- [Developer's Guide](#)
 - [The Application Framework](#)
 - [The Life-Cycle](#)
 - [Modules Overview](#)
 - [Starter Classes and Configuration](#)
 - [Querying](#)
 - [Logging](#)
 - [Threading](#)
 - [Interfacing With Platform-Specific Code](#)
 - [A Simple Game](#)
 - [Extending the Simple Game](#)
 - [File Handling](#)
 - [Networking](#)
 - [Preferences](#)
 - [Input Handling](#)
 - [Configuration & Querying](#)
 - [Mouse, Touch and Keyboard](#)
 - [Polling](#)
 - [Event Handling](#)
 - [Controllers](#)
 - [Gesture Detection](#)
 - [Simple Text Input](#)
 - [Accelerometer](#)
 - [Compass](#)
 - [Gyroscope](#)
 - [Vibrator](#)
 - [Cursor Visibility & Catching](#)
 - [Back and Menu Key Catching](#)
 - [On-Screen Keyboard](#)
 - [Memory Management](#)
 - [Audio](#)
 - [Sound Effects](#)
 - [Streaming Music](#)
 - [Playing PCM Audio](#)
 - [Recording PCM Audio](#)
 - [Graphics](#)

- [Querying & configuring graphics \(monitors, display modes, vsync\)](#)
- [Continuous & Non-Continuous Rendering](#)
- [Clearing the Screen](#)
- [Taking a Screenshot](#)
- [Profiling](#)
- [Viewports](#)
- [OpenGL ES Support](#)
 - [Configuration & Querying OpenGL ??](#)
 - [Direct Access ??](#)
 - [Utility Classes](#)
 - [Rendering Shapes](#)
 - [Textures & TextureRegions](#)
 - [Meshes](#)
 - [Shaders](#)
 - [Frame Buffer Objects](#)
- [2D Graphics](#)
 - [SpriteBatch, TextureRegions, and Sprites](#)
 - [2D Animation](#)
 - [Clipping, With the Use of ScissorStack](#)
 - [Orthographic Camera](#)
 - [Mapping Touch Coordinates ??](#)
 - [NinePatches](#)
 - [Bitmap Fonts](#)
 - [Distance Field Fonts](#)
 - [Color Markup Language](#)
 - [Using TextureAtlases](#)
 - [Pxmmaps](#)
 - [Packing Atlases Offline](#)
 - [Packing Atlases at Runtime](#)
 - [Texture Compression](#)
 - [2D ParticleEffects](#)
 - [Tile Maps](#)
 - [scene2d](#)
 - [scene2d.ui](#)
 - [Table](#)
 - [Skin](#)
 - [ImGui](#)
- [3D Graphics](#)
 - [Quick Start](#)
 - [Models](#)
 - [Material and Environment](#)
 - [ModelBatch](#)
 - [ModelCache](#)
 - [ModelBuilder, MeshBuilder and MeshPartBuilder](#)
 - [3D Animations and Skinning](#)

- [Importing Blender Models in libGDX](#)
- [3D Particle Effects](#)
- [Perspective Camera ??](#)
- [Picking ??](#)
- [Managing Your Assets](#)
- [Internationalization and Localization](#)
- [Utilities](#)
 - [Reading and Writing JSON](#)
 - [Reading and Writing XML](#)
 - [Collections](#)
 - [Reflection](#)
 - [jnigen](#)
- [Math Utilities](#)
 - [Interpolation](#)
 - [Vectors, Matrices, Quaternions](#)
 - [Circles, Planes, Rays, etc.](#)
 - [Path Interface and Splines](#)
 - [Bounding Volumes ??](#)
 - [Intersection & Overlap Testing ??](#)
- [Tools](#)
 - [Texture Packer](#)
 - [Hiero](#)
 - [2D Particle Editor](#)
 - [Path Editor](#)
- [Extensions](#)
 - [Artificial Intelligence](#)
 - [gdx-freetype](#)
 - [gdx-pay](#): cross-platform In-App-Purchasing API
 - [Physics](#)
 - [Box2D](#)
 - [Bullet Physics](#)
 - [Setup](#)
 - [Using the Wrapper](#)
 - [Using Models](#)
 - [Contact Callbacks](#)
 - [Custom Classes](#)
 - [Debugging](#)
- [Using libGDX With Other JVM Languages](#)
 - [Using libGDX With Clojure](#)
 - [Using libGDX With Kotlin](#)
 - [Using libGDX With Python](#)
 - [Using libGDX With Scala](#)
- [Third Party Services](#)
 - [AdMob in libGDX](#)
 - [Airpush in libGDX](#)

- [Swarm in libGDX](#)
- [NextPeer in libGDX](#)
- [Google Play Games Services in libGDX](#)
- [ProGuard/DexGuard and libGDX](#)
- [Excelsior JET and libGDX](#)
- Articles
 - [Getting Help](#)
 - [External Tutorials](#)
 - [Bundling a JRE](#)
 - [Deploying as an Applet](#)
 - [Getting ready for #libGDXJAM](#)
 - [Coordinate systems](#)
 - [Updating Your libGDX Version](#)
 - [Adding Extensions and 3rd Party Libraries](#)
 - [Publishing Your Own Extensions Via the Setup Application](#)
 - [Improving Your Gradle Workflow](#)
 - [Creating Asset Project in Eclipse](#)

Clone this wiki locally

`https://github.com/libgdx/libgdx.wiki.git`

