Java™ Platform, Micro Edition

Part 8 – Mobile 3D Graphics

Disclaimer

- **These slides are provided free of charge at http://www.symbianresources.com and are used during Java ME courses at the University of Applied Sciences in Hagenberg, Austria at the Mobile Computing department (http://www.fh-ooe.at/mc)**
- **Respecting the copyright laws, you are allowed to use them:**
  - **for your own, personal, non-commercial use**
  - **in the academic environment**
- **In all other cases (e.g. for commercial training), please contact andreas.jakl@fh-hagenberg.at**
- The correctness of the contents of these materials cannot be guaranteed. Andreas Jakl is not liable for incorrect information or damage that may arise from using the materials.
- This document contains copyright materials which are proprietary to Sun or various mobile device manufacturers, including Nokia, SonyEricsson and Motorola. Sun, Sun Microsystems, the Sun Logo and the Java™ Platform, Micro Edition are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Agenda

- Mobile 3D – Overview
- JSR 184
- Scene graph
- Your first m3g file with Blender
- Display, load and modify the 3D scene
- Objects and materials

Mobile 3D

Introduction

3D Java Game Examples

Brothers in Arms: Earned in Blood (Gameloft)

Tornado Mania! 3D (Digital Chocolate)

Planet Riders 3D (Fishlabs)

Massive Snowboarding 3D (Gameloft)

Blades & Magic 3D (Fishlabs)

For comparison: Current C++ / Open GL ES based N-Gage games (Symbian OS)
(This is what is currently possible with mid-/higher class mobile phones)

Star Wars: The Force Unleashed (Universomo / LucasArts)

One (Digital Legends)

Hooked On: Creatures of the Deep (Infinite Dreams)

Benchmark Your Phone

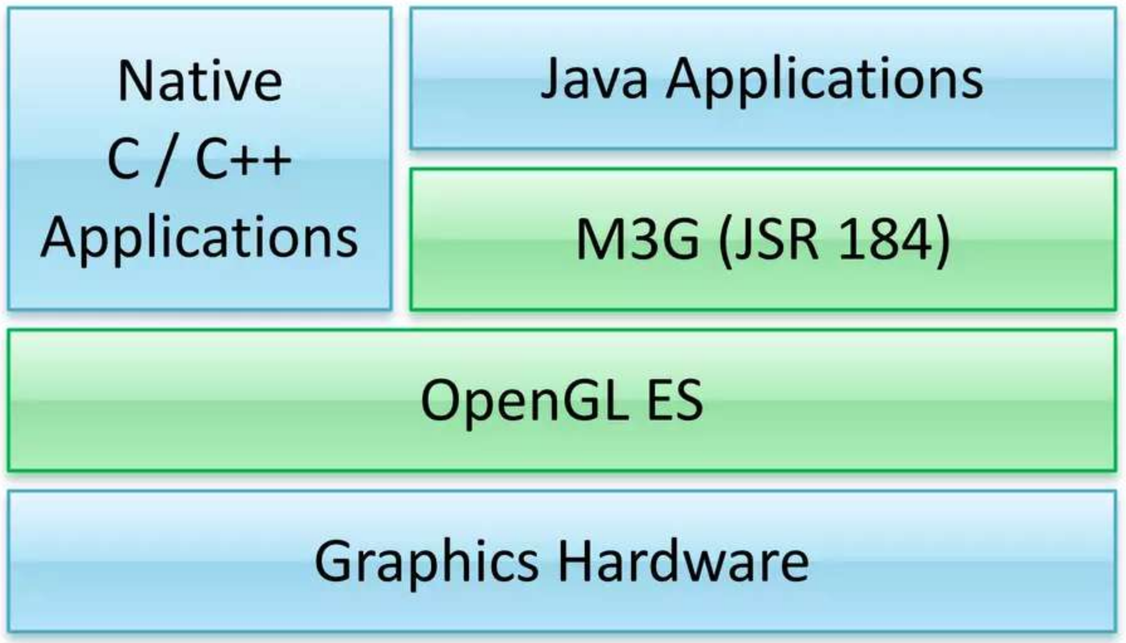
- Futuremark **SPMarkJava** JSR 184
  http://www.futuremark.com/products/spmark/spmarkjavajsr184/
- **JBenchmark** 3D/HD:
  http://www.jbenchmark.com/

3D APIs for Java ME

- **Mobile 3D Graphics API (JSR 184)**
  - Java based 3D graphics library
  - Supports immediate and retained mode (low and high level)
  - Optional package
- **Open GL ES (JSR 239)**
  - Provide Java bindings to the Open GL ES native 3D graphics library
  - OpenGL ES can also be used in native code (Symbian OS / C++)
- **Mascot Capsule**
  - Like JSR 184, additionally supported by e.g. SonyEricsson
- *(Java 3D from Java SE is too bloated for mobile phones: ~40 MB)*

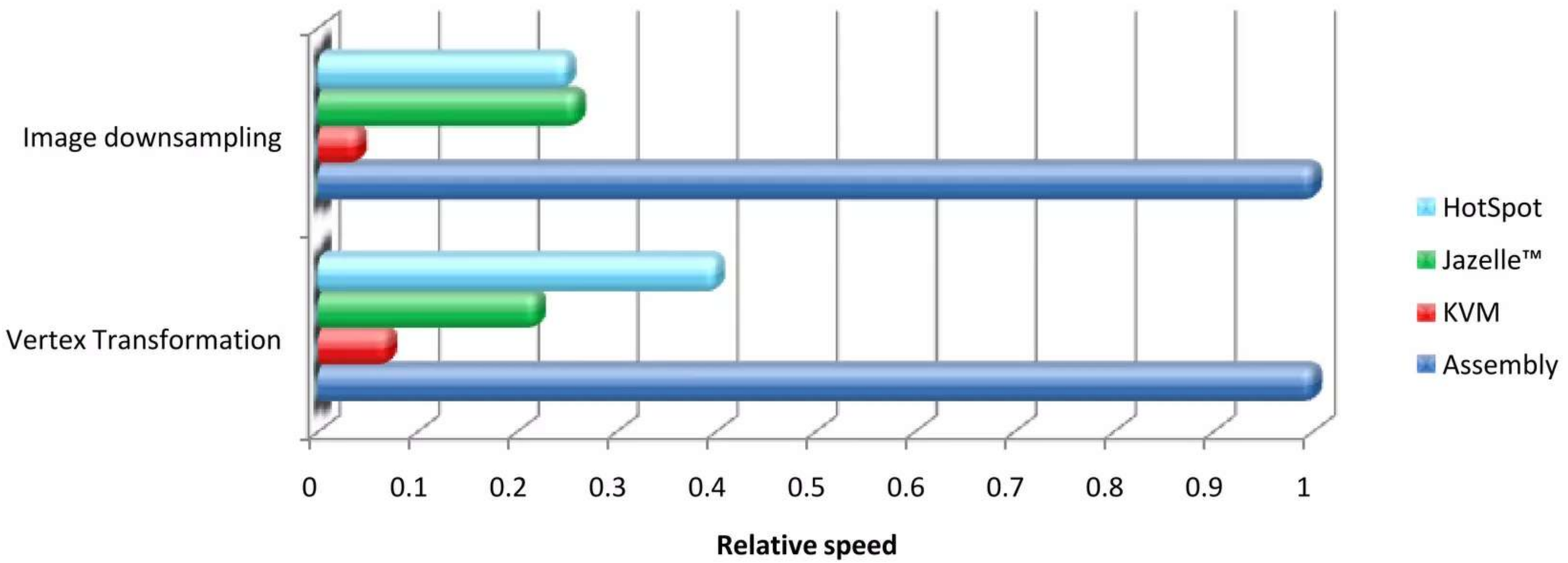
3D APIs

- M3G builds on the feature set of OpenGL ES
- Both APIs are designed concurrently



Performance

- Java is slow on mobile phones
- KVM most widely used today
- Nokia (and others) migrating to HotSpot VM from Sun
- But HotSpot takes a lot of RAM → problematic in real life



Benchmarked on an ARM926EJ-S processor with hand-optimized Java and assembly code
Diagram from Tomi Aarnio (see sources slide at the end)

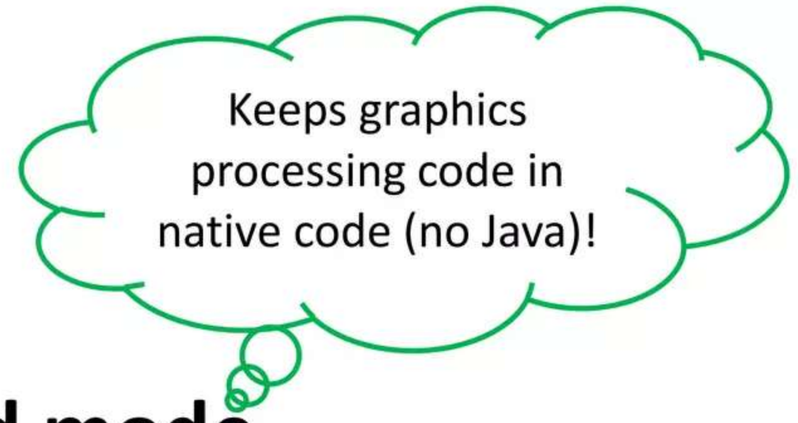
Get started

The Basics of JSR 184

JSR 184

- **Hardware independent**
  - Uses newer hardware + 3D accelerators if present
- **Separate code from content**
  - m3g file format contains models, animation, appearance, textures, ...
  - Control logic in source code
- **Tightly integrated** with LCDUI (from MIDP, requires floating point support of CLDC 1.1+)
- Built in **high level & low level API**
  - Scene graphs, keyframe animation, bones, ...

Modes

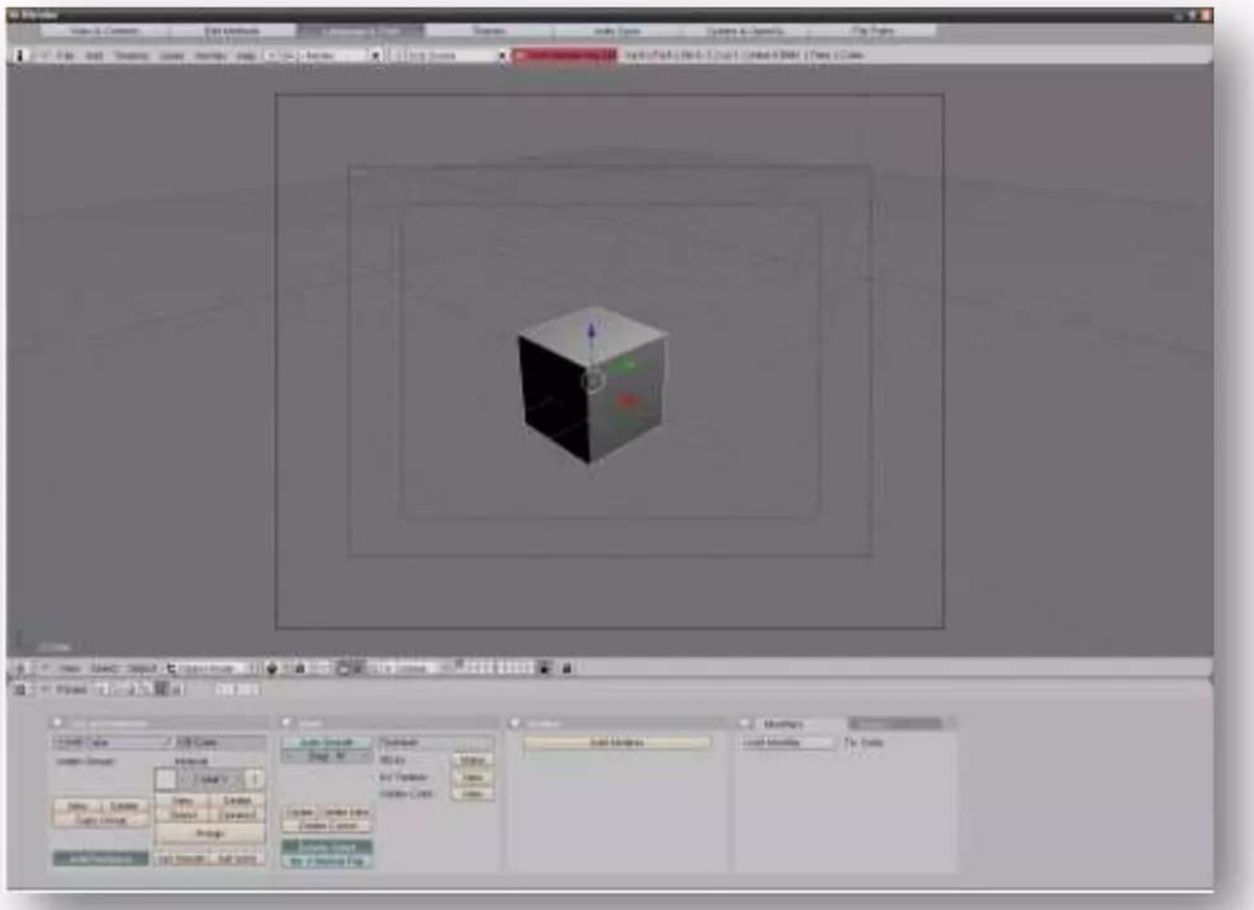


- **Immediate Mode**
  - Create objects programmatically
  - Uses triangles as primitives
  - Low level
- **Retained mode**
  - Create objects in 3D app
  - Scene graphs

Mixing is possible
(e.g. insert immediate object into a scene graph)

Tools

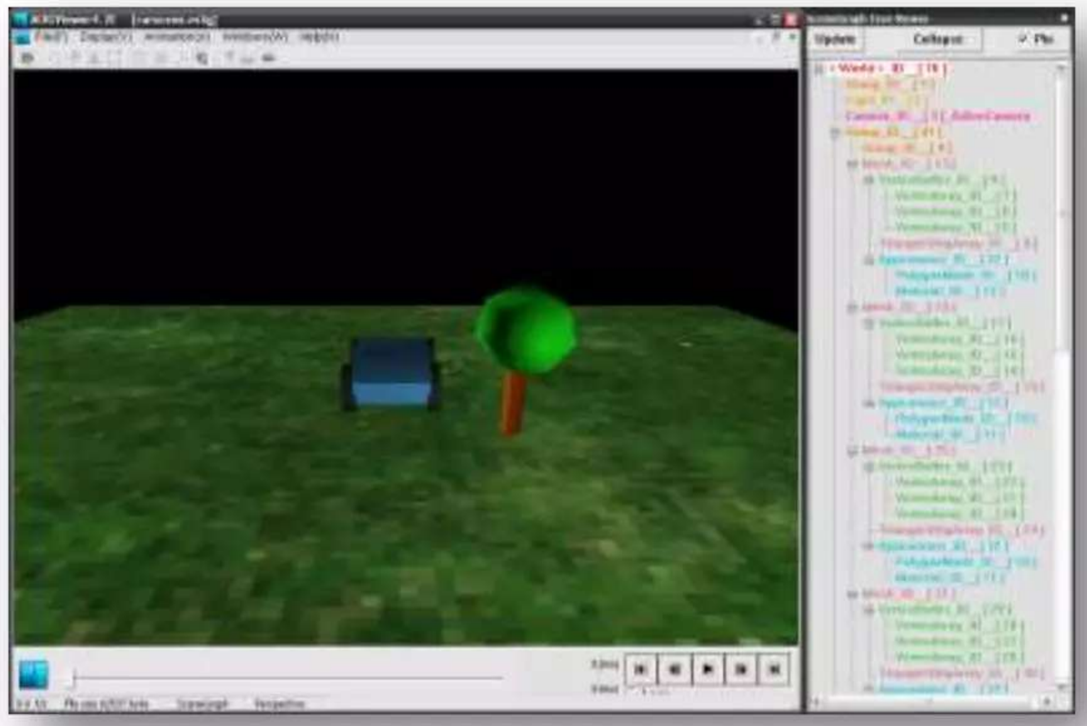
- **Blender** (Freeware, http://www.blender.org/)
  - m3g exporter plug-in: http://www.nelson-games.de/bl2m3g/default.html
  - Blender: Powerful, but difficult to learn UI
  - Written in Python
- **Commercial**
  - 3DS Max (integrated)
  - Maya
  - Lightwave



JSR 184 – Synchronous Structure

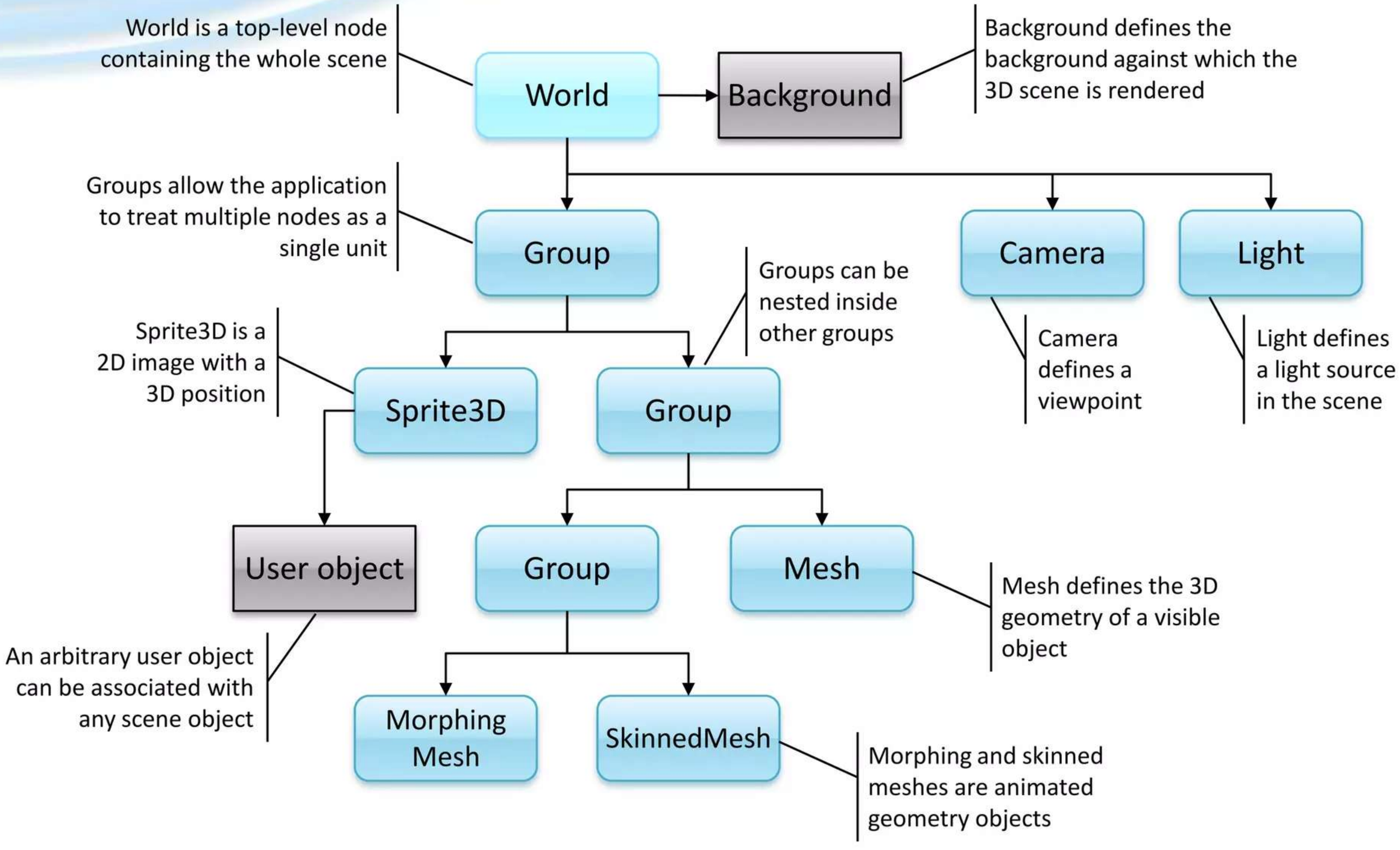
- **All APIs are synchronous**
  - Calls return when completed
  - Create own thread for loading larger scenes!
- **Structure**
  - No callbacks (interfaces, events, abstract methods)
  - Do not override any methods

m3g File Format



- Compact **binary format**
- Stores **complete scene graph** in file
  - Objects are serialized
  - Includes camera(s), lightning and objects
- Can be **compressed**
- **Viewer:** http://www.mascotcapsule.com/toolkit/m3g/en/index.php

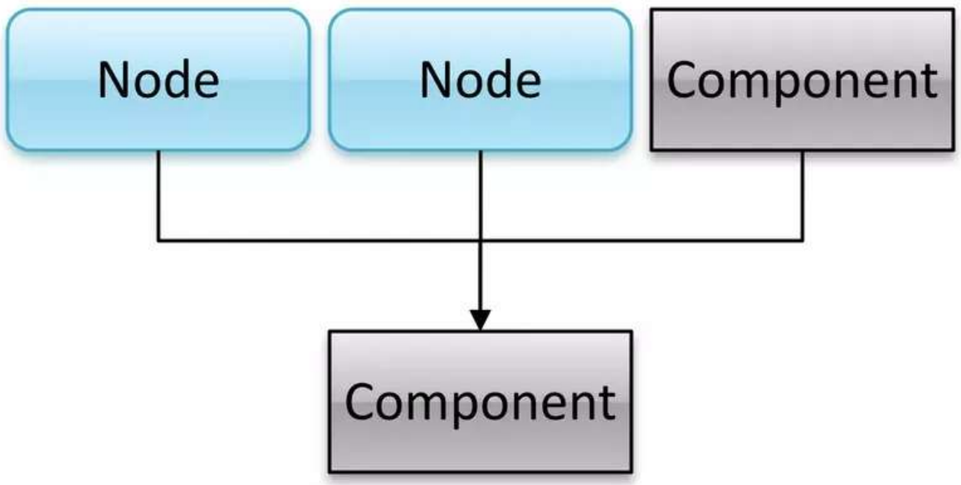
Scene Graph

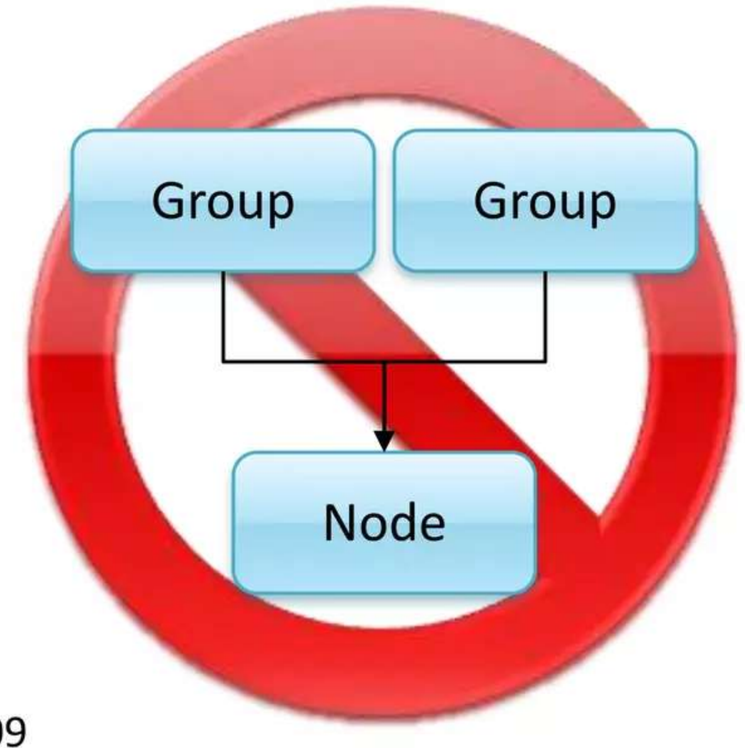


Based on the scene graph image from the JSR 184 API documentation

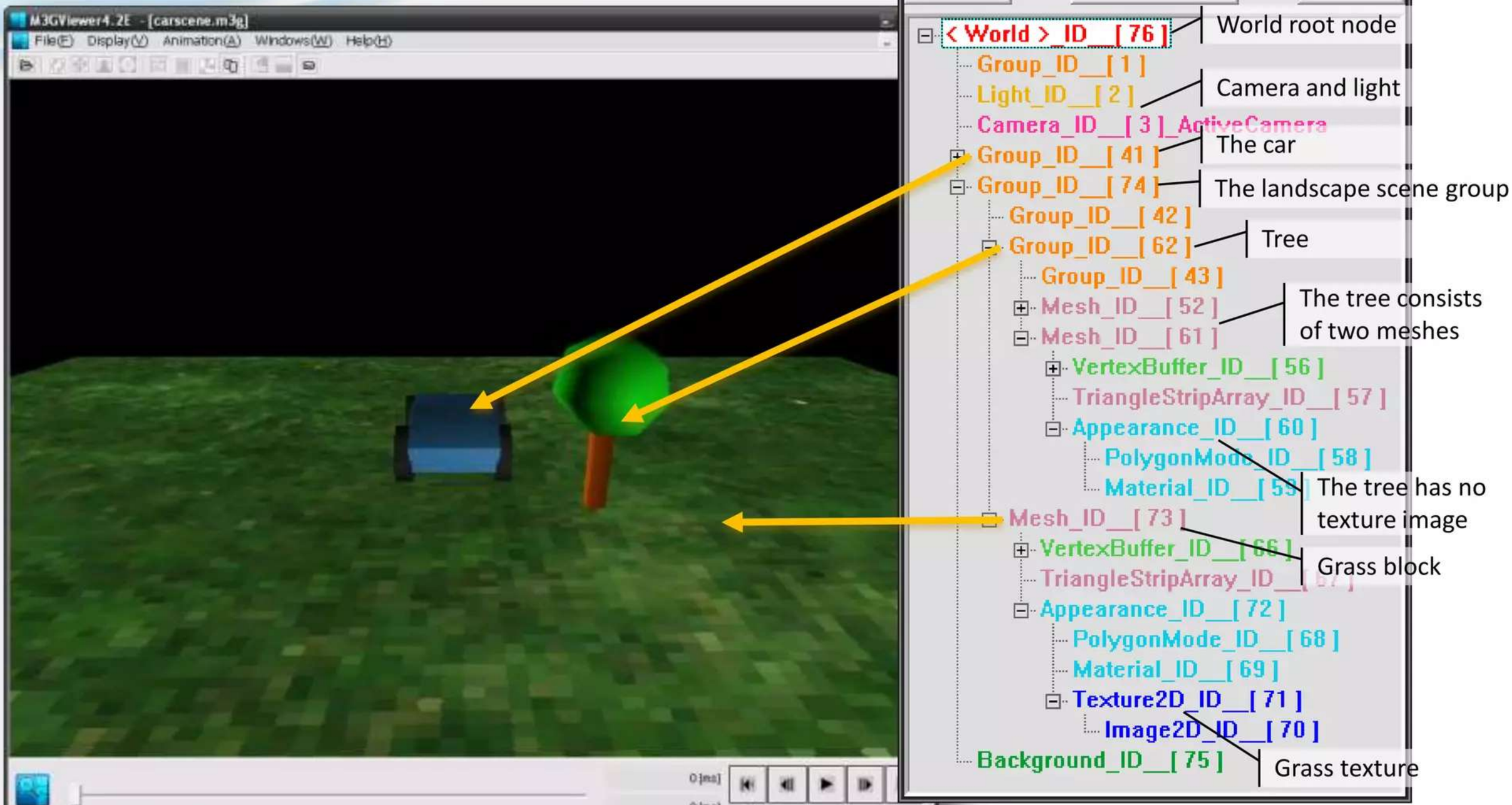
Graph vs. Tree

- Scene graph has a **tree structure**
  - **Node** can belong to at most **one group** at a time
  - But **components** (VertexArrays, textures, ...) can be **shared**
  - No cyclic structures allowed



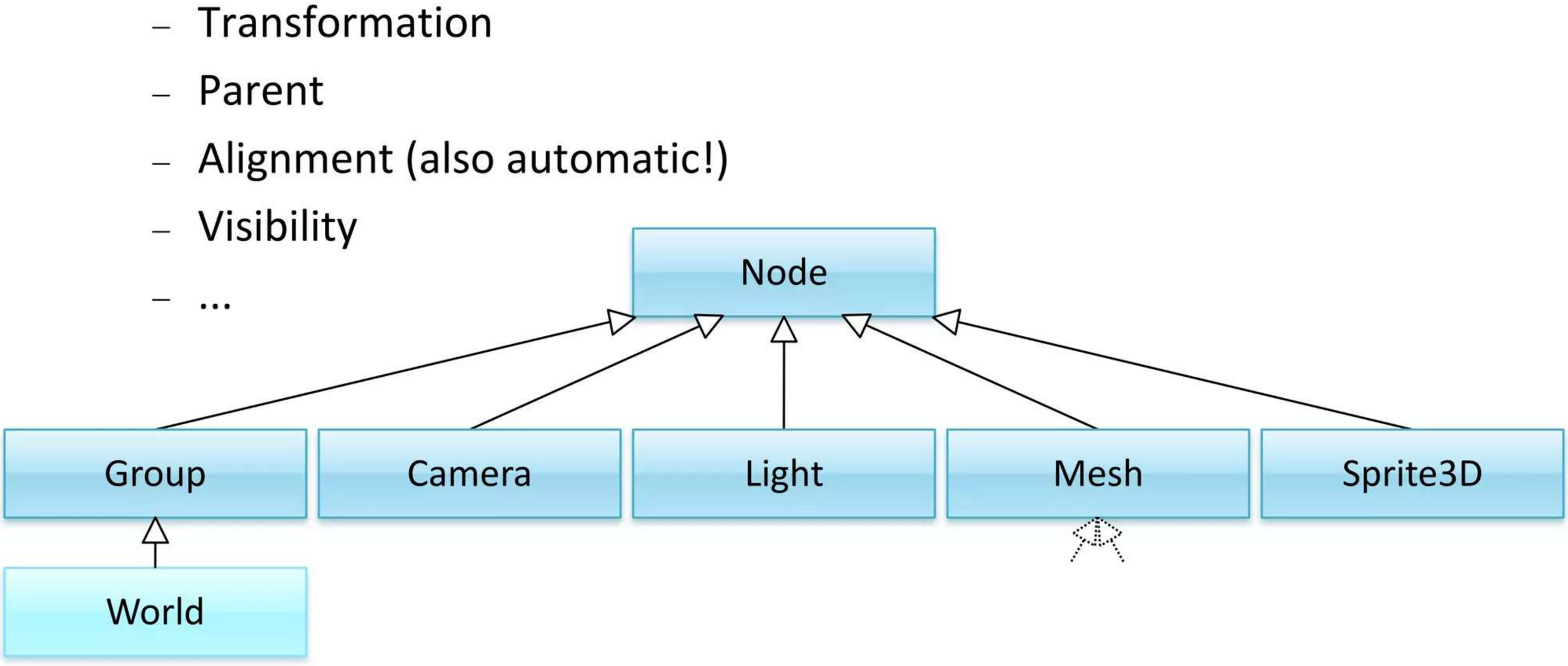


Scene Graph – Example



Node

- Abstract base class for all scene graph nodes
- Contains information about:
  - Transformation
  - Parent
  - Alignment (also automatic!)
  - Visibility
  - ...



Key Classes

Graphics3D	3D graphics context Handles all rendering
Loader	Utility to load m3g and png files (entire scene graphs or single objects)
World	Root node of the scene graph

Graphics3D

Graphics3D

- **Stores global state**
  - Rendering target, viewport, depth buffer, back buffer
  - Camera, light sources
  - Rendering quality hints (Antialiasing, dithering, true color rendering)
- **Each node has its own local state**
  - Geometry & appearance (material, textures, ...)
  - Transformation relative to the parent or world

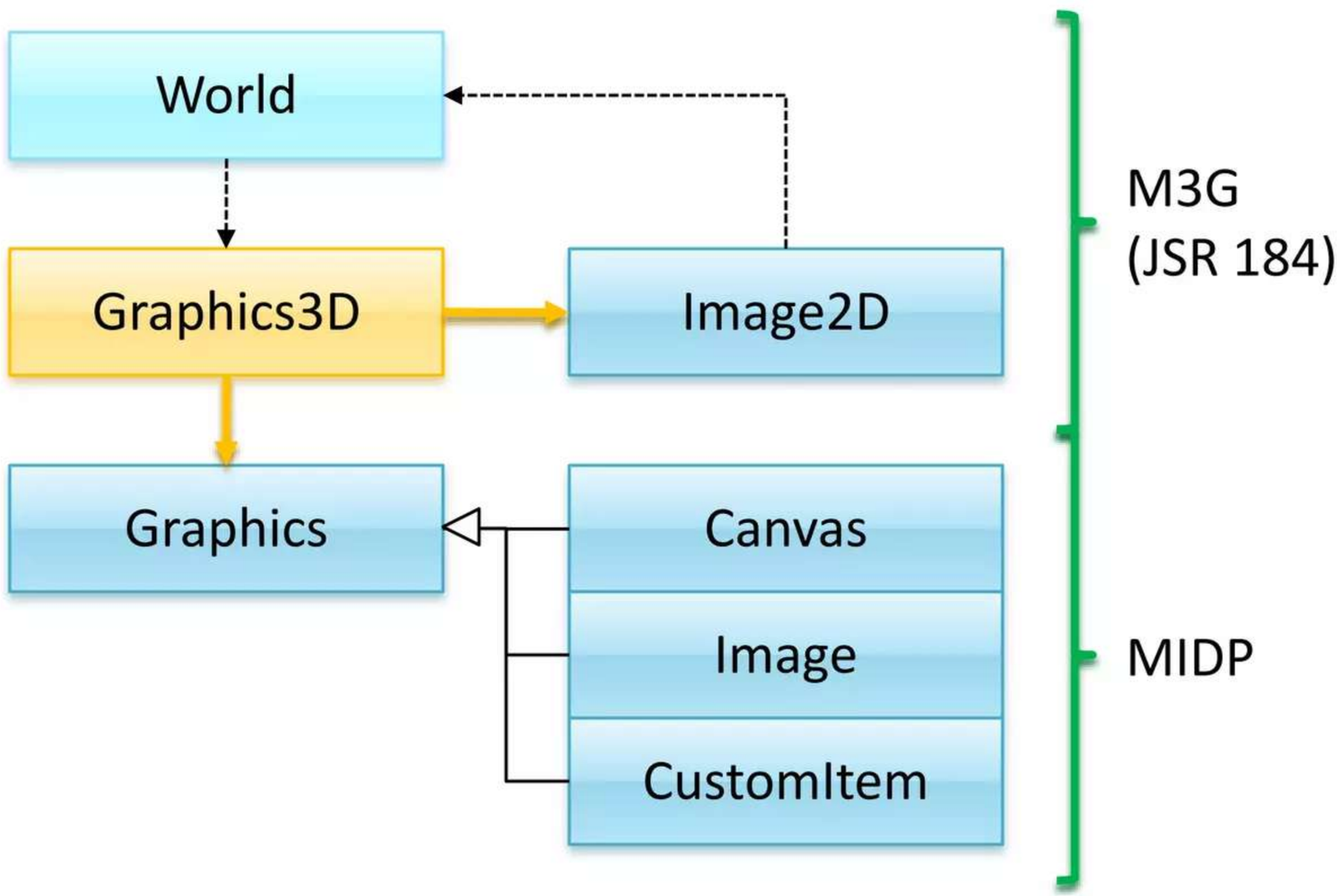
Graphics3D – Rendering Modes

- **Retained mode**
  - Render entire world (scenegraph)
  - Uses lights and camera nodes from the world
- **Immediate mode**
  - Render scene graph nodes or submesh
  - Uses lights and camera from Graphics3D object

Graphics3D – Render Targets

- Graphics3D can render to any Graphics object or Image2D

  → possible to render to textures (eg. environment mapping)



Graphics3D – Example

- Target has to be bound and released
- Do not modify the target from the outside in between

```
public class MyCanvas extends Canvas
{
   Graphics3D myG3D = Graphics3D.getInstance();

   public void paint(Graphics g) {
   try {
      myG3D.bindTarget(g);
      // ... update the scene ...
      // ... render the scene ...
   } finally {
      myG3D.releaseTarget();
   }
}
```

World Node

- **Top level node** for a scene graph
- **Contains**
  - Other nodes that compose the scene
  - Background
  - Camera, Light, Fog
- Requires an **active camera** for rendering

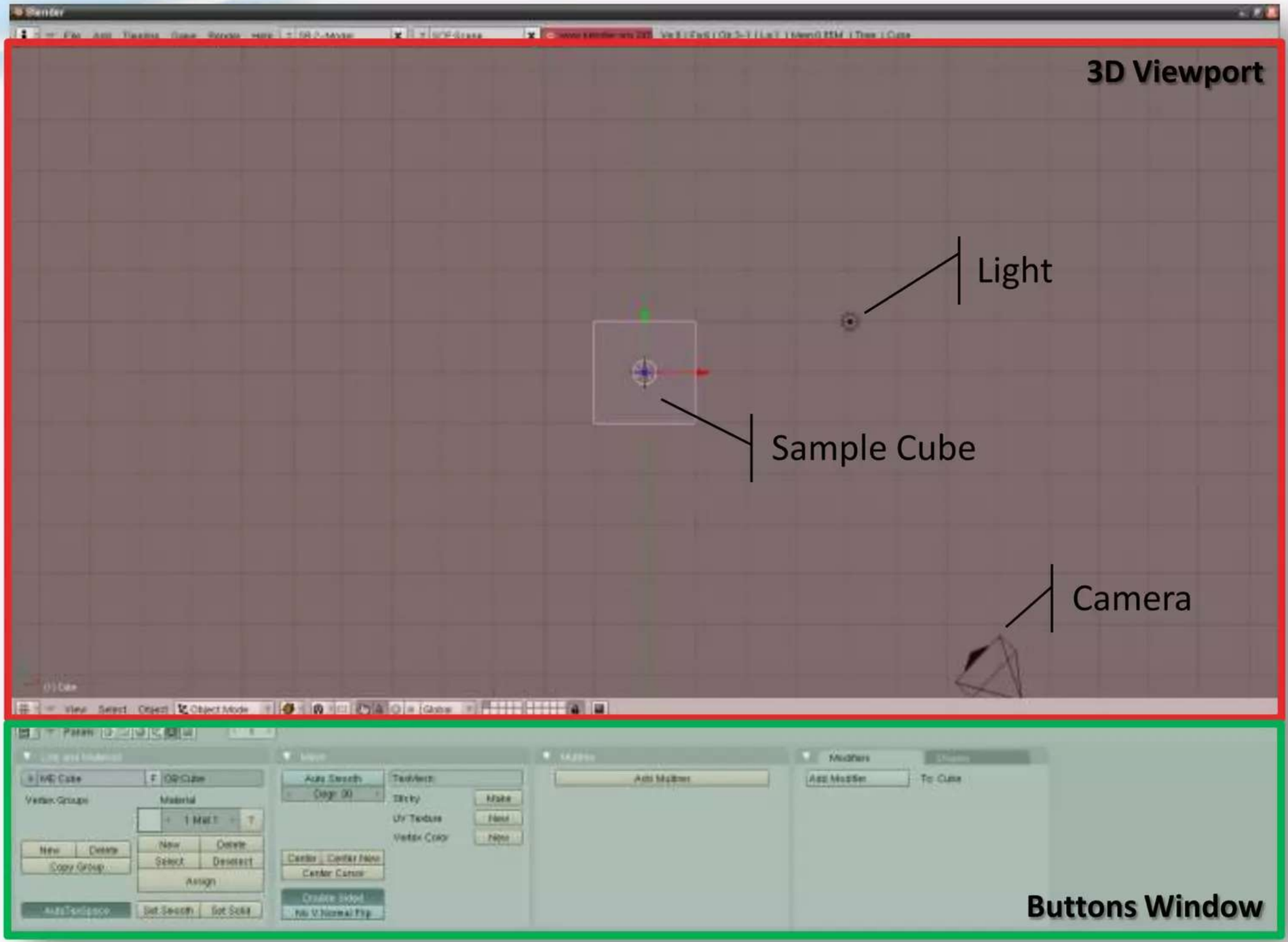
Example – Goal

- Create an m3g file with Blender
- Load and display the file in a MIDlet
- Rotate the object

The basics of

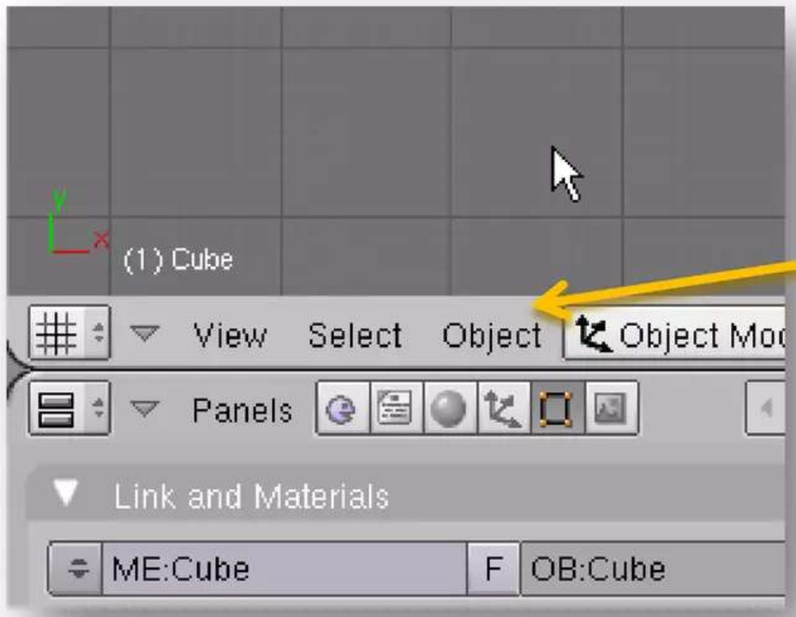
Blender

Blender – Interface

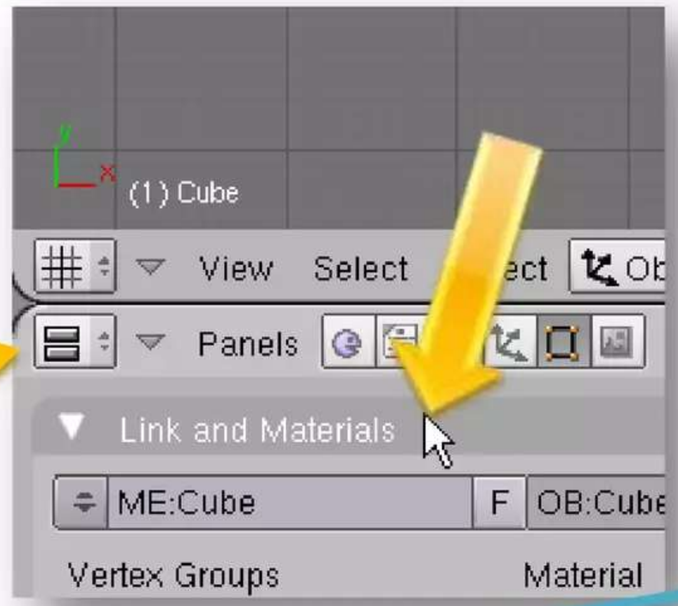


Blender – Windows

The highlighted window receives keyboard inputs:



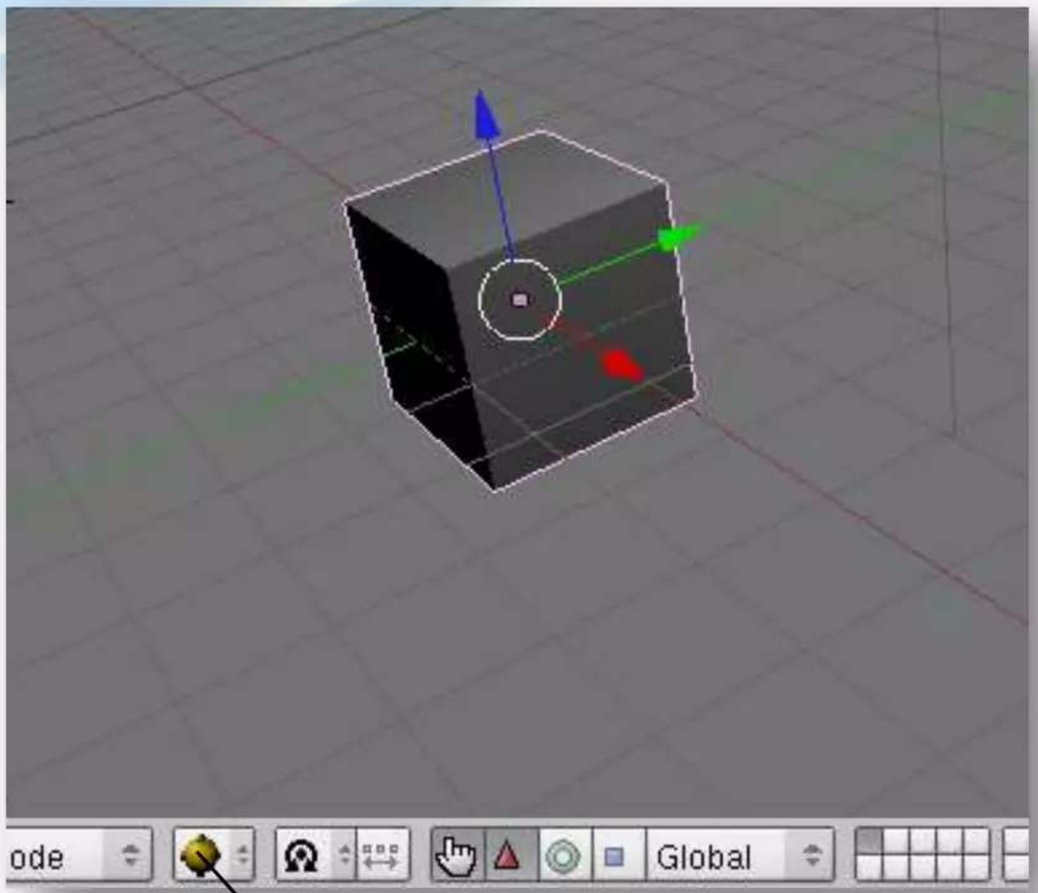
Highlighted (slightly brighter colour)



Change the window type:

... you can split, join and resize windows and drag mini windows (with the ▼)

Blender – 3D Viewport



Choose viewport shading
(more options than Z)

Click + hold **middle mouse button**
→ Rotate viewport

Shift + middle mouse button
→ Pan view (move sideways)

Mouse wheel
→ Zoom

Z-key
→ Toggle wireframe or solid mode

NUM5 (with NUM LOCK on)
→ Toggle Ortographic / Perspective

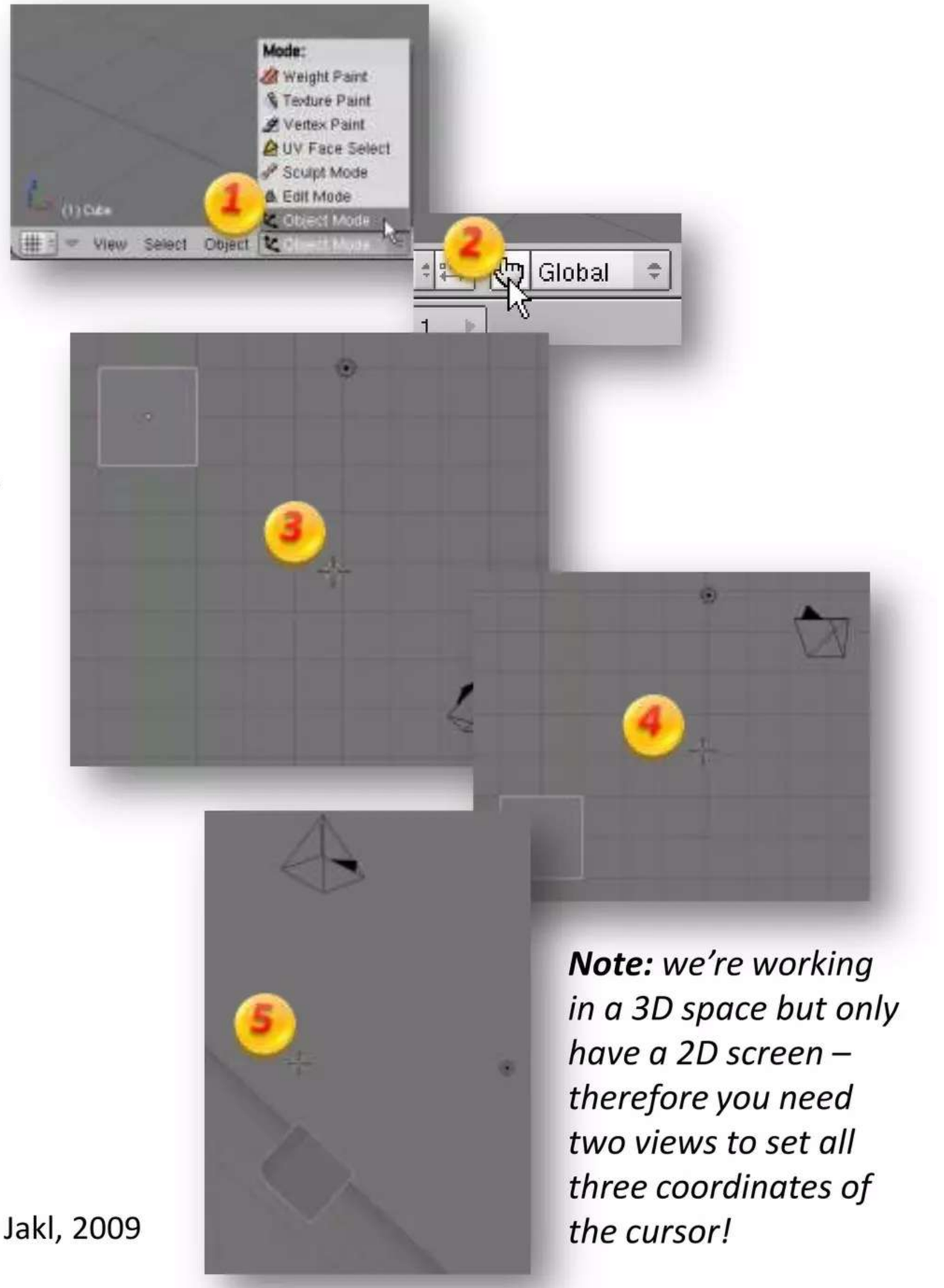
NUM refers to the number pad – if you pressed the numbers above the normal keys instead, press 1 to return to the normal view (for viewing layer 1)

NUM0 (with NUM LOCK on)
→ Camera view (MMB will exit it)

The 3D Cursor

Task: place the 3D cursor  between the camera and the cube

1. Make sure you are in **object mode** (press **TAB** to toggle)
2. Disable the "**3d transform manipulator**" to make sure you can move the cursor without selecting the cube by accident
3. Hit **NUM7** to get to the top view
4. Click with **LMB** between cube and camera
5. Choose different view (**NUM1** – front view or **NUM3** – side view)
6. Click between cube and camera with **LMB**
7. Rotate the view to see if it was positioned correctly

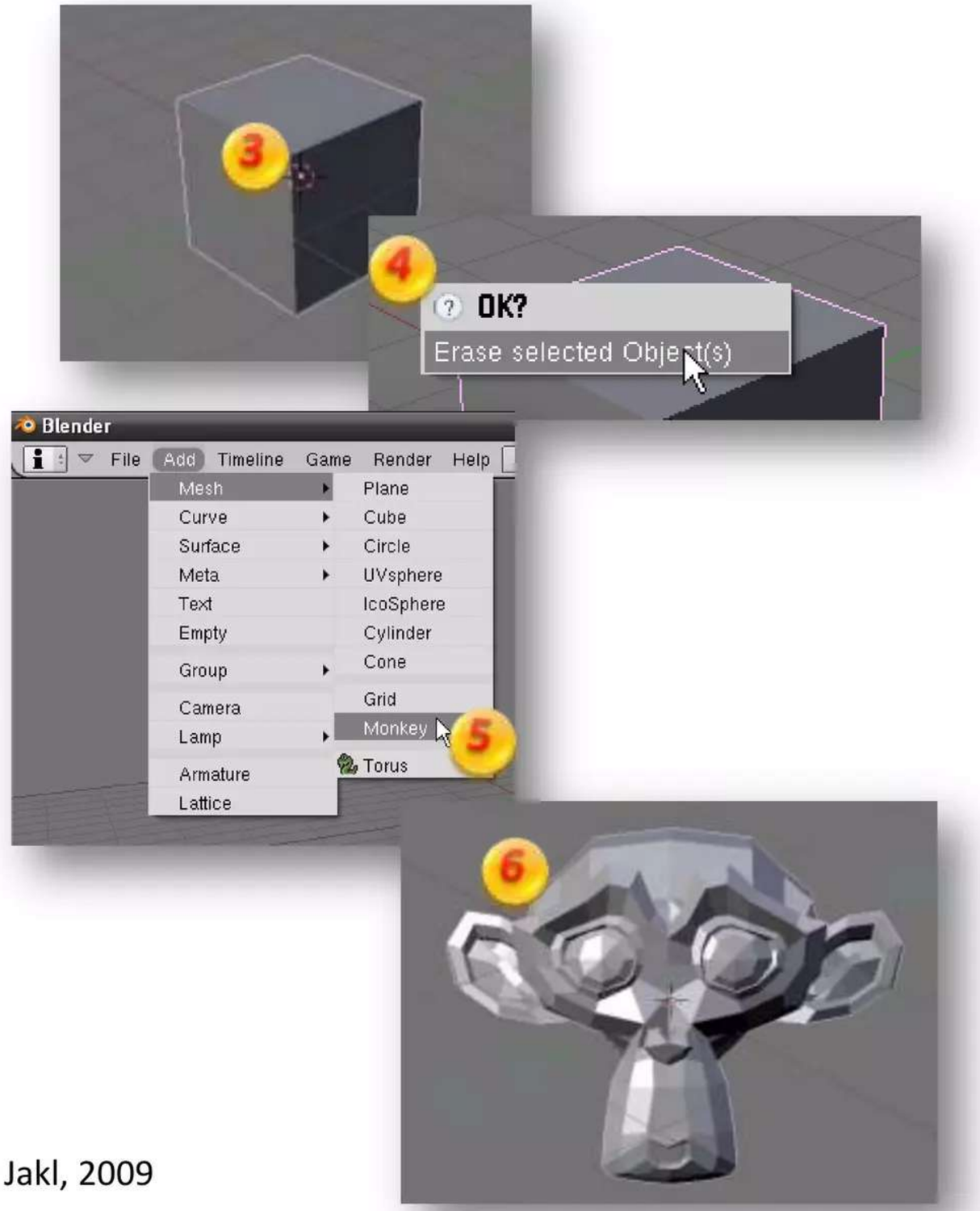


Note: *we're working in a 3D space but only have a 2D screen – therefore you need two views to set all three coordinates of the cursor!*

Deleting and Adding Objects

Task: replace the cube with a monkey

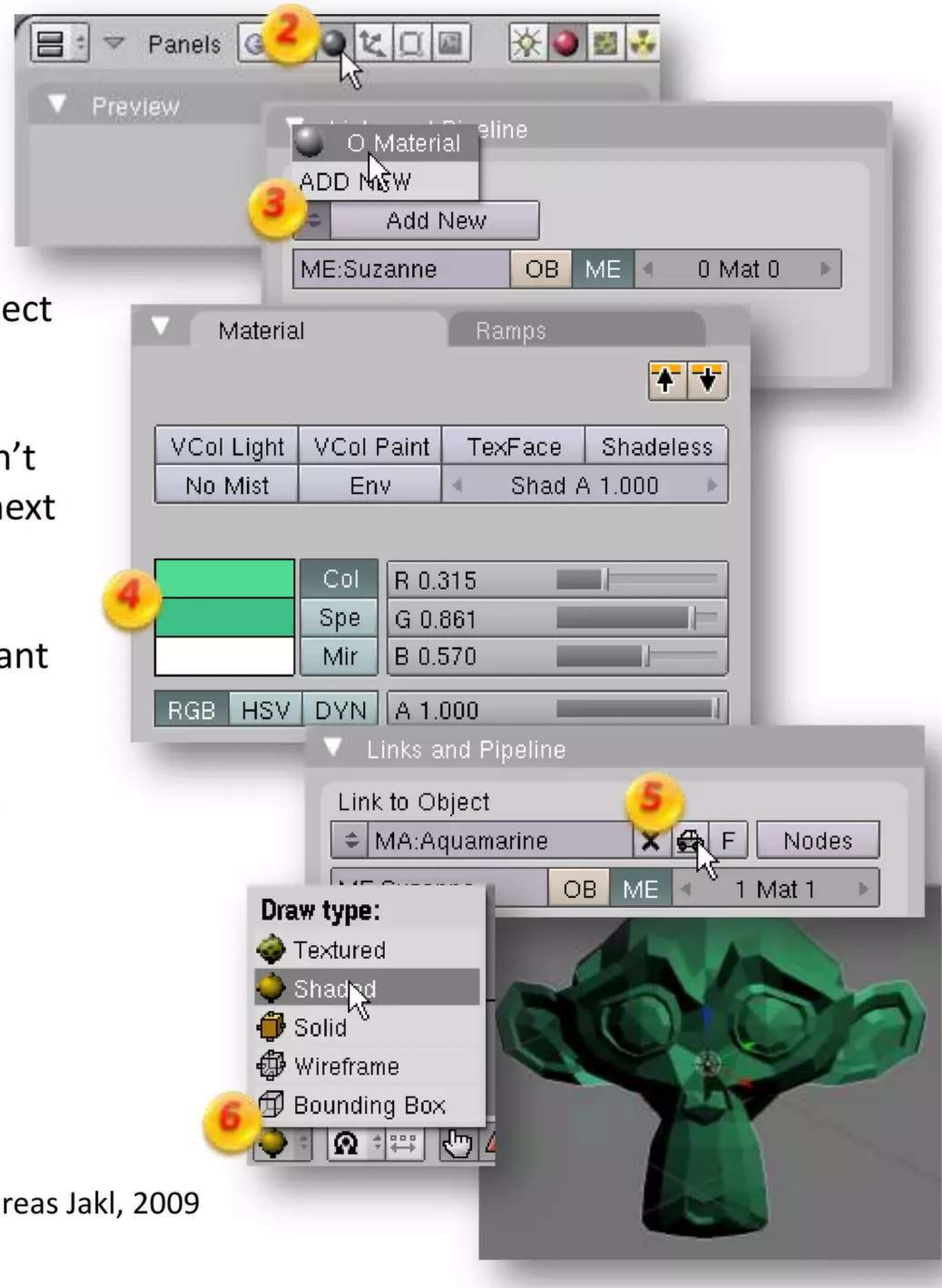
1. Set “Object Mode”; switch off “3d transform manipulator” (see previous slide)
2. Move the cursor to the center (**Shift+CKEY**)
3. **RMB** on the cube to select it
4. **DELKEY** to delete the object. Confirm in the prompt window (“Erase Selected”)
5. Use the Add menu to add a new object. Here: **Add → Mesh → Monkey**
6. Blender automatically switches to edit mode. Go to object mode (**TAB**), press **CKEY** to center the cursor on the screen, **ZKEY** to toggle solid and wireframe mode. Zoom in (**Mouse wheel**)



Material

Task: change the material of the monkey

1. Select the monkey object (**RMB**); set “Object Mode”
2. Press the **shading button**
3. In contrast to the cube, the monkey doesn’t have a default material. Click the button next to “**Add New**” and choose “**O Material**” (same that was originally on the cube)
4. In the “Material” tab, set any color you want for “**Col**” and “**Spe**” (specular color, for highlights)
5. Press the button with the **small car** in the “Links and Pipeline” tab for an automatic material name
6. Set the draw type of the 3D Viewport to “**Shaded**” for a more accurate preview



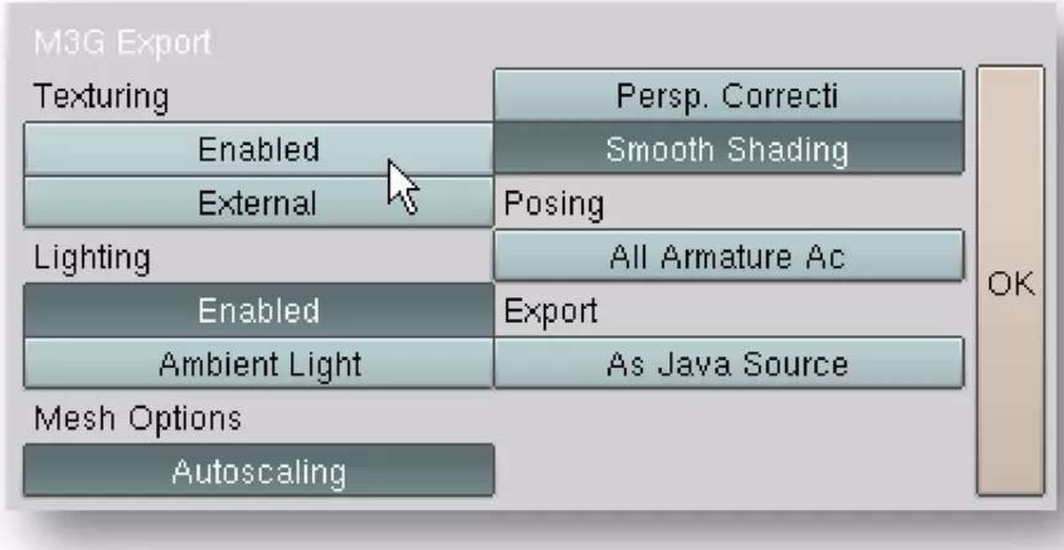
IDs and m3g Export

Task: to access the nodes from the m3g file in source code, they need IDs.

1. Set “Object Mode”
2. Select the monkey with the **RMB**
3. Add **#01** at the end of the object name to give it the ID 1 in the m3g file
4. Do the same for the light **(#02**) and the camera (**#03**)

Task: export the m3g file (requires the m3g exporter plug-in)

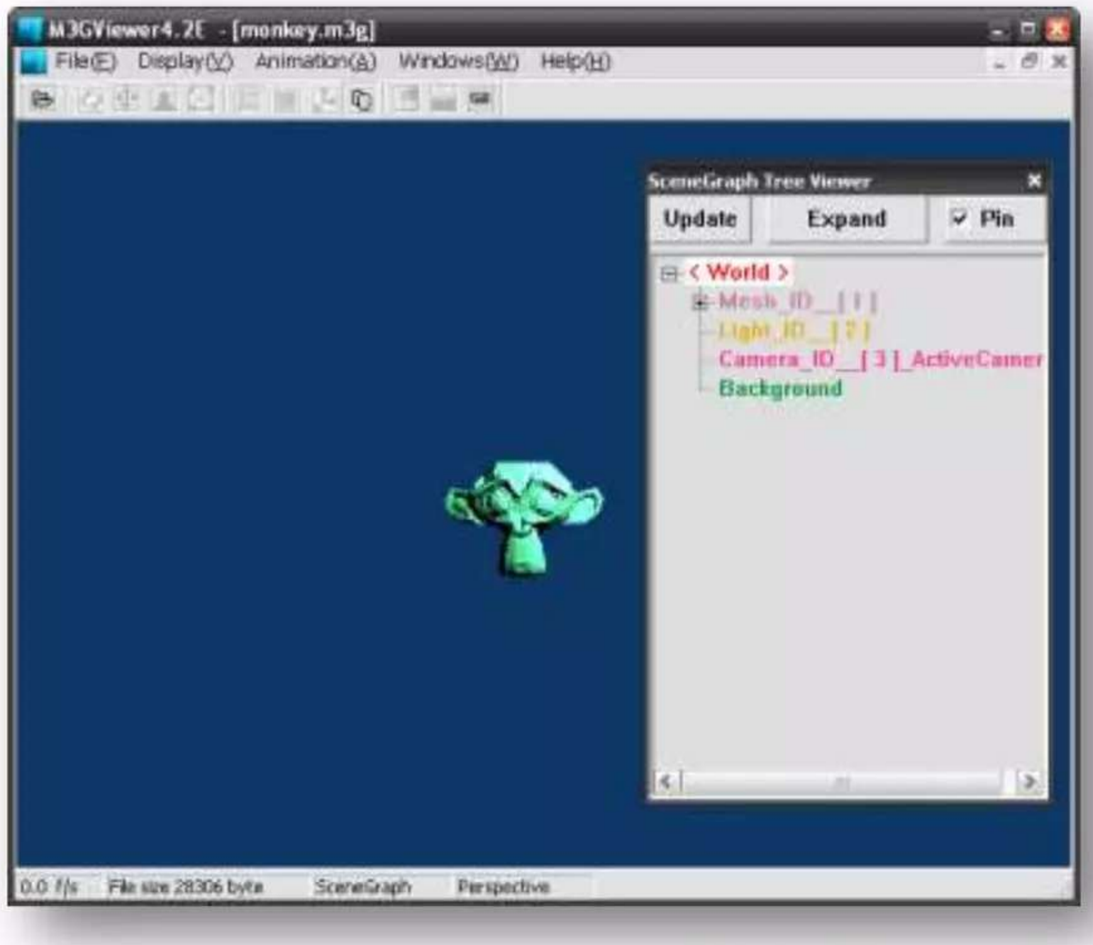
1. File → Export → M3G
2. You can safely disable texturing



m3g File

Open the m3g file in the m3g viewer* and choose Display → Scene Graph View

Our IDs have been saved in the file



* http://www.mascotcapsule.com/toolkit/m3g/en/index.php

Switching to the Java side

Display the 3D scene

Loading the m3g File

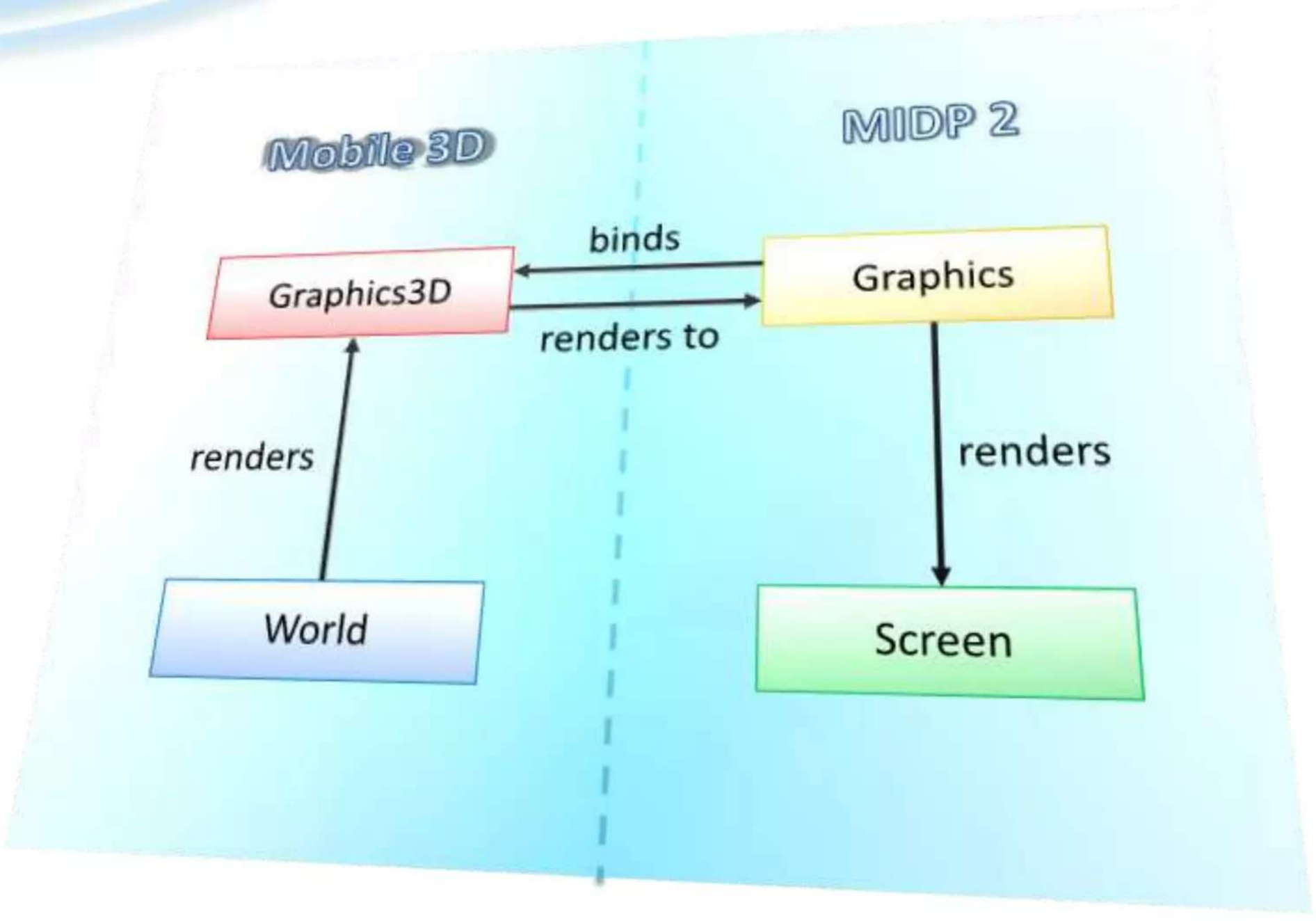
- Create a simple MIDlet and Canvas-class (sample start project is provided)
- Add the monkey.m3g to the root of the .jar
- Init code snippet of the Canvas:

```
Object3D[] roots = null;
try {
   // Load the m3g file
   // The loader will return all root level objects of the file,
   // which are not referenced by any other objects.
   // Always state an absolute path ("/")!
   roots = Loader.load("/monkey.m3g");
} catch (IOException ex) {
   // couldn't open the file, or invalid data in the file
   ex.printStackTrace();
}
// Usually the world node is the only and first node.
// If loading unknown files, you should check it though.
// Save the world node to an instance variable
for (int i = 0; i < roots.length; ++i) {
   if (roots[i] instanceof World) {
      iWorld = (World) roots[i];
   }
}
```

Game loop in a GameCanvas

```
public void run() {
   // Get the singleton Graphics3D instance that is associated with this midlet.
   Graphics3D g3d = Graphics3D.getInstance();
   // Measure the time that has passed since the prev. frame
   long start, elapsed = 0; int time = 0;
   while (iIsActive) {
      start = System.currentTimeMillis();
      try {
         // Bind the 3D graphics context to the given MIDP Graphics object.
         g3d.bindTarget(getGraphics());
         // Update the world [...]
         iWorld.animate(time);            // Animate the world
         // Render the view from the active camera
         g3d.render(iWorld);
      } finally {
         // Release the graphics context
         g3d.releaseTarget();
      }
      flushGraphics();        // Flush the rendered image to the screen
      // Give other threads a chance to run
      Thread.sleep(20);     // This sample omits try and catch
      elapsed = System.currentTimeMillis() - start;
      time += elapsed;      // Time that has passed since the last frame
   }
}
```

MIDP Relationship



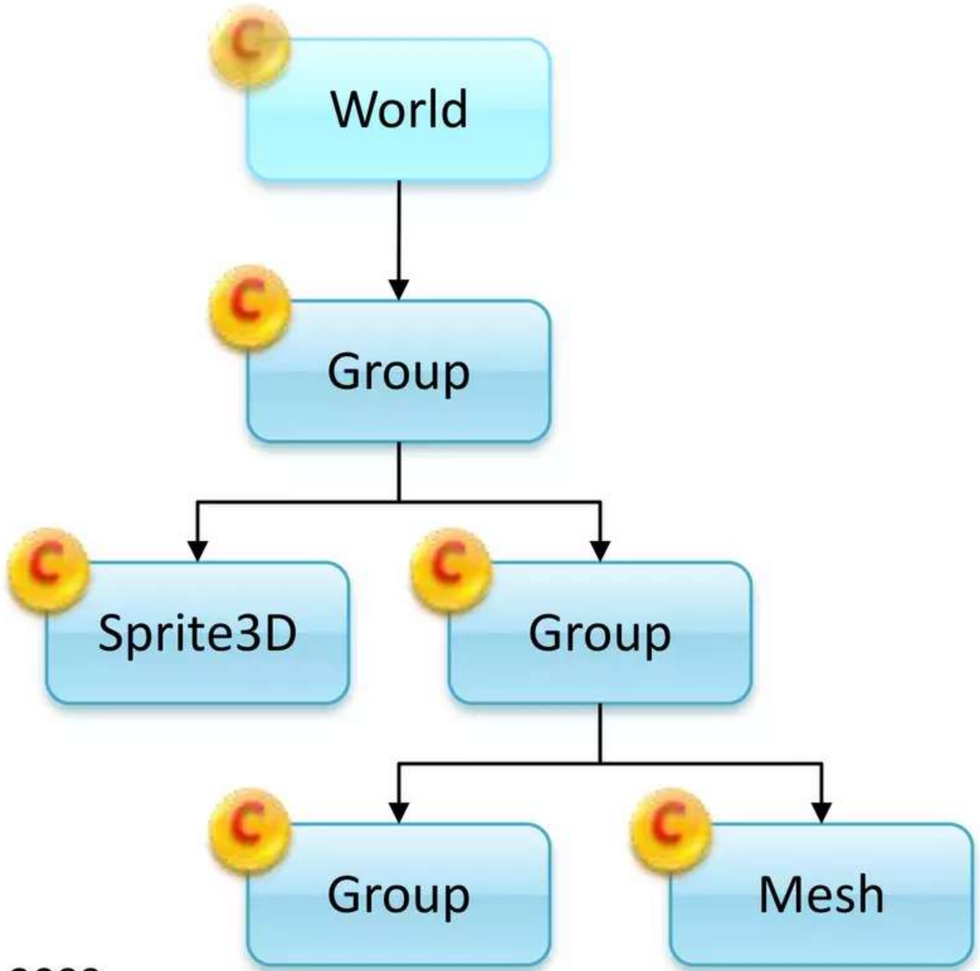
The Camera

- The camera was too far away in the Blender scene
- Move it towards the object after loading the world:

```
Camera cam = iWorld.getActiveCamera();
cam.setTranslation(-5.0f, 5.0f, -4.0f);
```

Node Transformations

- Each node: local coordinate system
- Node transformation: *from* this node *to* the parent node (ignored for root)
  - Translation T
  - Rotation R
  - Non-uniform scale S
  - Generic matrix M
- Composite C = TRSM



Modifying Objects

- Rotate the monkey

```
private static final int ID_MONKEY = 1;

// Find searches through all children of this node
Node monkey = (Node) iWorld.find(ID_MONKEY);
switch (getKeyStates())
{
   case LEFT_PRESSED:
      // Parameters: angle, x / y / z component of
      // the rotation axis. Here: all on z axis
      monkey.postRotate(5.0f, 0.0f, 0.0f, -1.0f);
      break;
   case RIGHT_PRESSED:
      monkey.postRotate(5.0f, 0.0f, 0.0f, 1.0f);
      break;
}
```

What is actually rendered?

Objects and Materials

Textures

- **Add visual richness**
  - Backgrounds
  - Surface structure
  - Sprites for “cheating” (no complex 3D models)

Character images copyright David Dang
http://www.chi-3d.co.uk/

Sprites in “Doom” by id Software

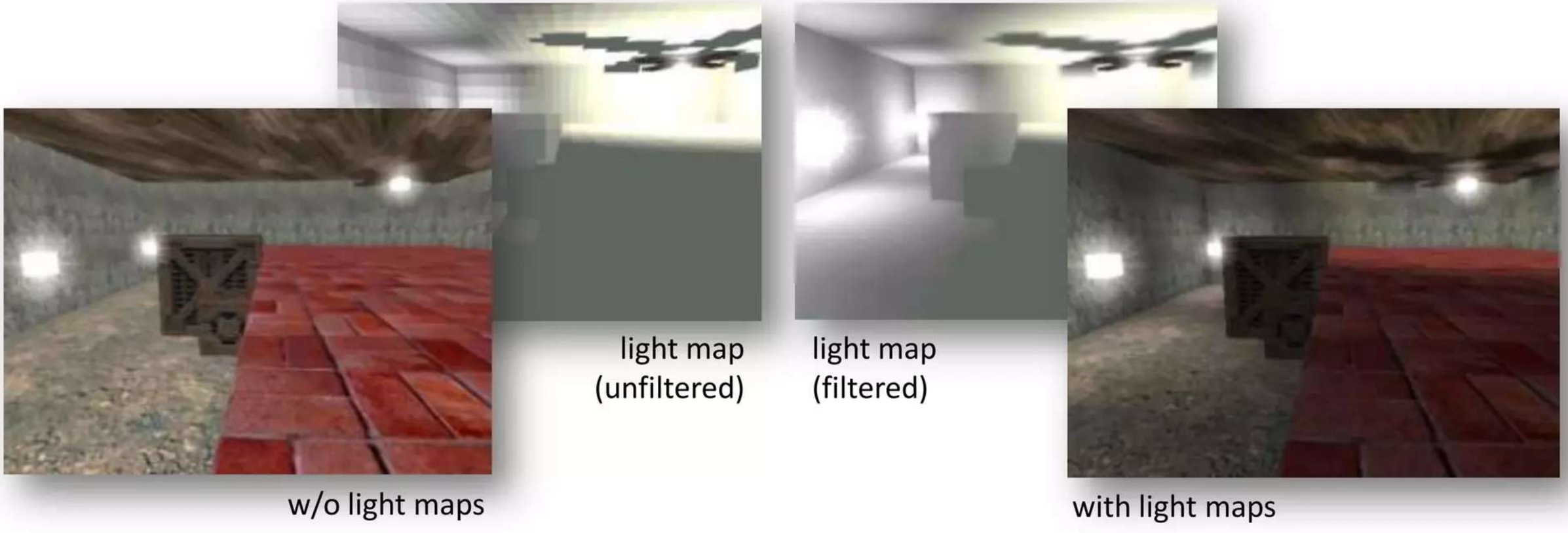
Perspective Correction

- **Perspective correction**
  - Expensive, avoid if possible
  - But still better than a lot more polygons
  - Nokia: 2% fixed overhead, 20% in the worst case

Perspective correction
http://en.wikipedia.org/wiki/Texture_mapping

Light Maps

- **Lightning**
  - Use light maps instead of real light
  - Pre-calculate light for non-textured (colored) objects: vertex coloring



Images from lecture slides of Stephen Chenney (see sources at the end)

Mobile Textures II

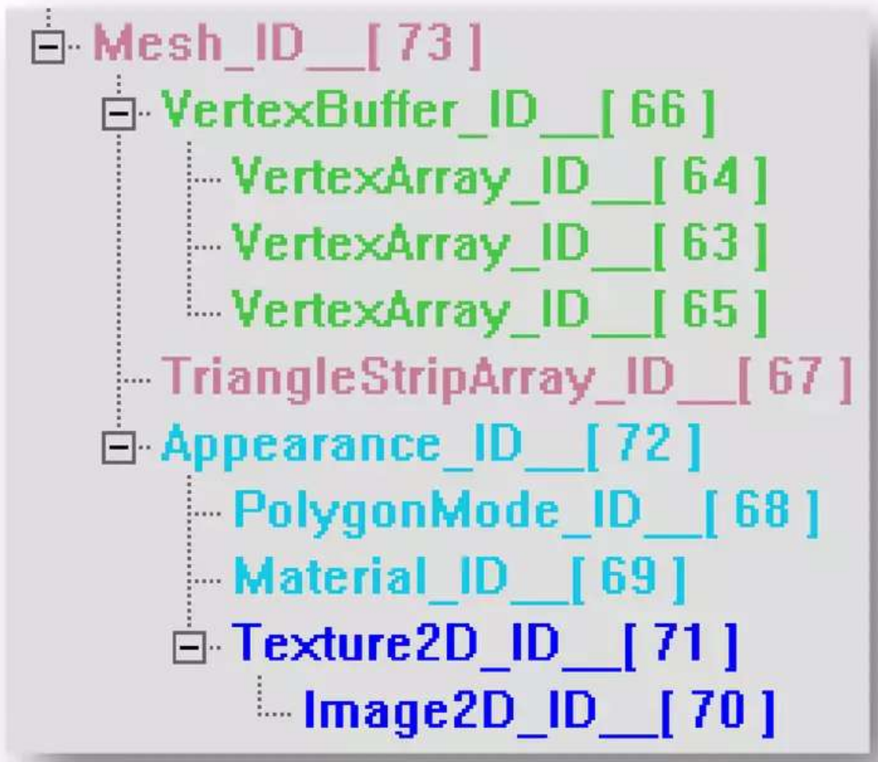
- **Backgrounds**
  - Use background images, no skybox or real 3D
- **Sprites**
  - Much faster than real geometry or textured quads
  - But too much overhead for particle effects
- **Images**
  - Load through Loader class (Image2D) – going through MIDP Image wastes memory

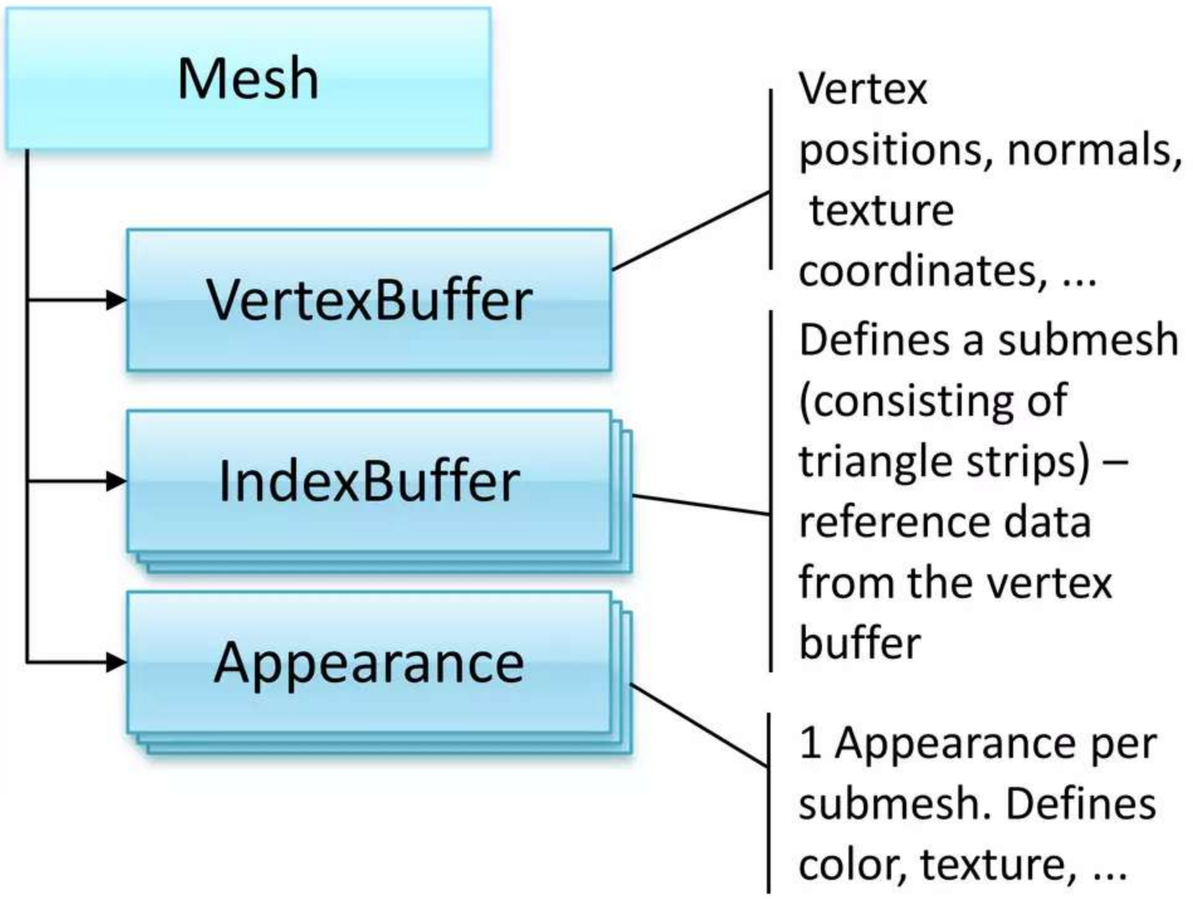
Sprites in a 3D scene
Playman World Soccer (Mr.Goodliving)

3D Sprites can be useful for 2D games as well (rotation!)
Tower Bloxx (Digital Chocolate)

Mesh

- How is the object defined? (vertices, material, ...)





Sprite3D

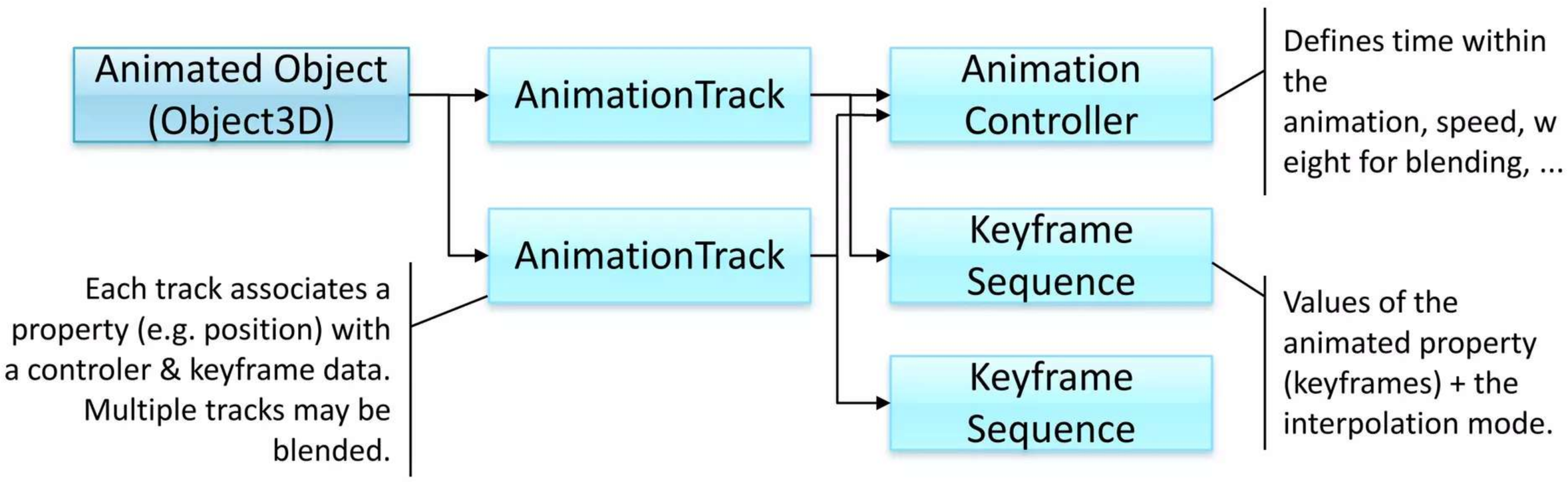
- **2D image with 3D position**
- Fast, but functionally restricted alternative to textured geometry
  - **Scaled mode:** billboards, trees, ...
  - **Unscaled mode:** text labels, icons, ...

Short overview of

Animation

Animations

- Defined in the m3g file
- Automatically played corresponding to the time by animate() method



KeyframeSequence

Keyframe
Sequence

- **Keyframe:** time & property value at that time
- **Sequence:**
  - Can store multiple keyframes
  - Several interpolation methods
  - Can be looping

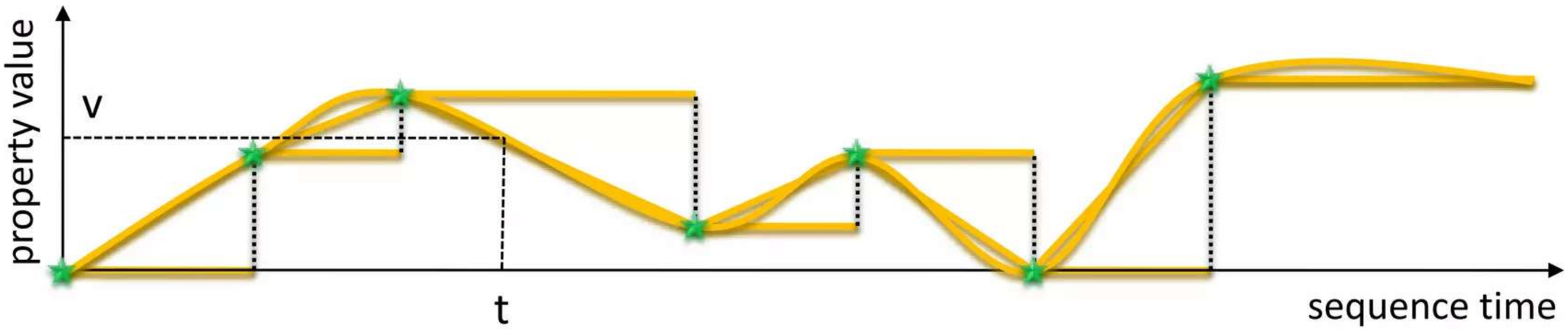


Diagram based on Sean Ellis, Superscape

KeyframeSequence

Keyframe
Sequence

- **Keyframe:** time & property value at that time
- **Sequence:**
  - Can store multiple keyframes
  - Several interpolation methods
  - Can be looping

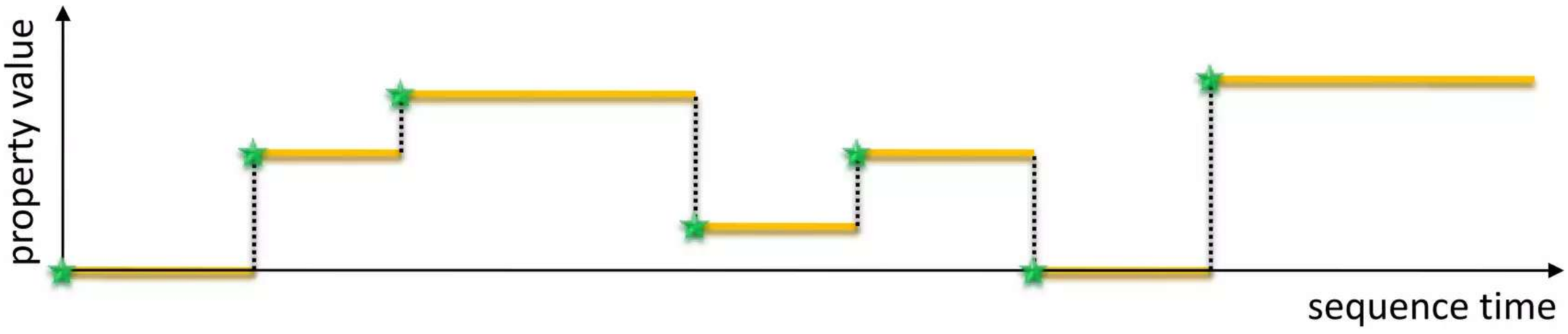


Diagram based on Sean Ellis, Superscape

KeyframeSequence

Keyframe
Sequence

- **Keyframe:** time & property value at that time
- **Sequence:**
  - Can store multiple keyframes
  - Several interpolation methods
  - Can be looping

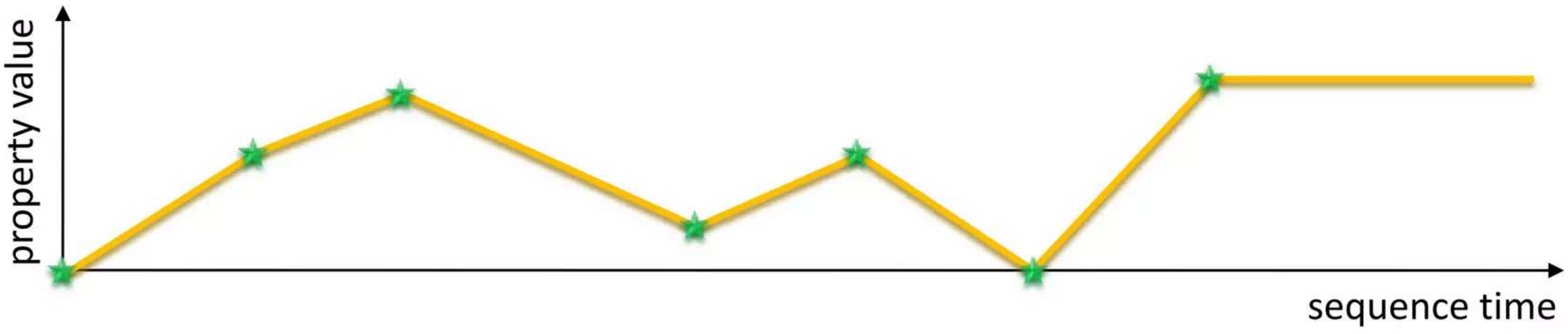


Diagram based on Sean Ellis, Superscape

KeyframeSequence

Keyframe
Sequence

- **Keyframe:** time & property value at that time
- **Sequence:**
  - Can store multiple keyframes
  - Several interpolation methods
  - Can be looping

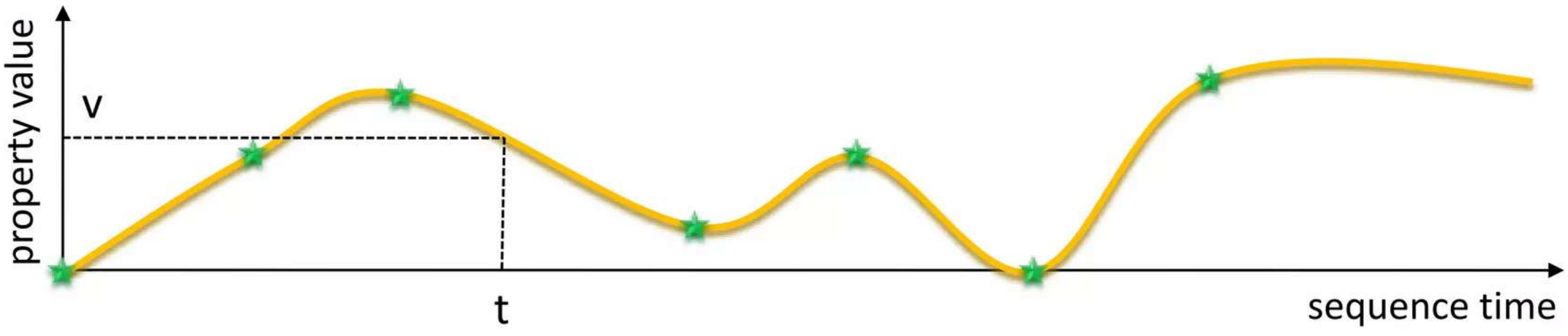


Diagram based on Sean Ellis, Superscape

AnimationController

Animation Controller

- Controls position, speed and weight of an animation sequence
- Usually controls multiple AnimationTracks (properties) at the same time
- Defines mapping from world time to sequence time

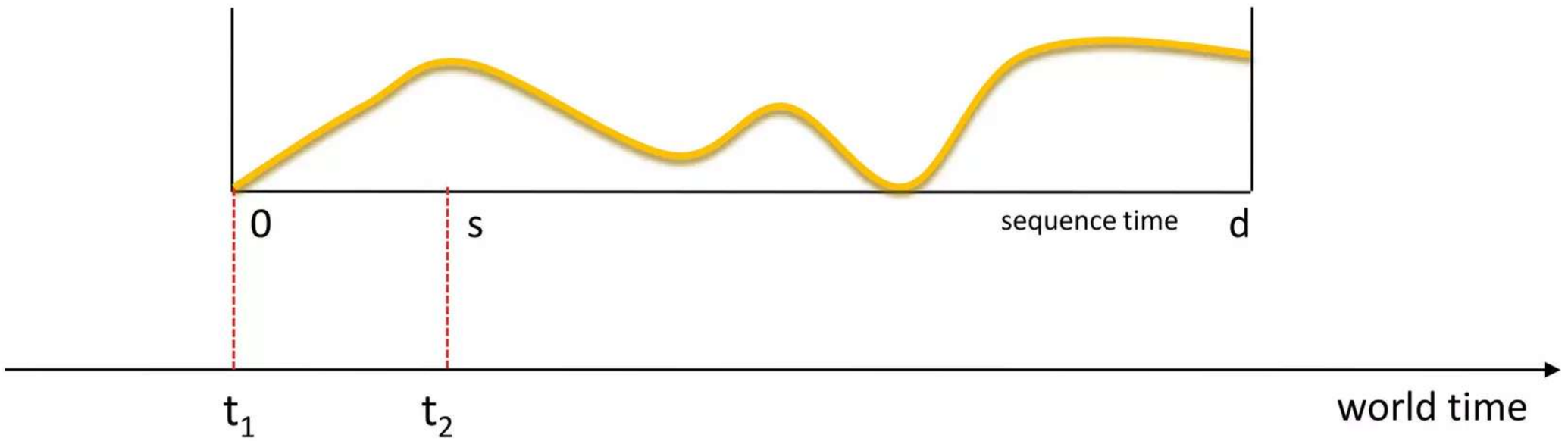
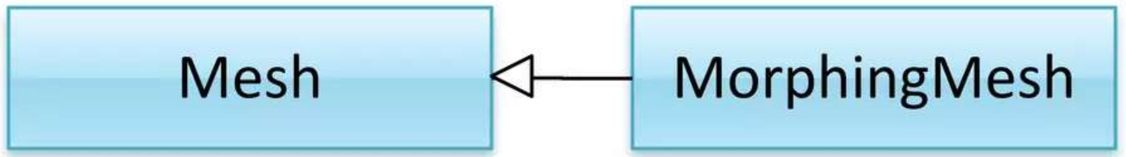


Diagram based on Sean Ellis, Superscape

Unfortunately, the animations are not exported to the pdf

Morphing



Base

Target 1
eyes closed

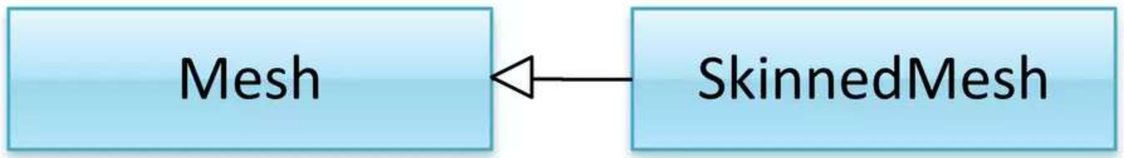
Target 2
mouth closed

Animate eyes and mouth independently

Images from Aarnio and Pulli (see sources at the end)

Unfortunately, the animations are not exported to the pdf

Skinning



No skinning

Local skinning
one bone per vertex

Smooth skinning
two bones per vertex

Images from Aarnio and Pulli *(see sources at the end)*

Summary

- **M3G enables real-time 3D on mobile phones**
  - Works both with software and hardware 3D
  - Supported by millions of devices
- **Mixture of low- and high level**
  - Allows easy creation of scenes
  - But still gives the developer full control
- **Flexible architecture**
  - Based on OpenGL ES features
  - Defines a convenient file format

Sources

- Tomi Aarnio and Kari Pulli: *Advanced Game Development with the Mobile 3D Graphics API*. JavaOne Conference, 2004.
- Andrew Davison: *Special Topics in Info. Eng.: Introduction to J2ME Programming*. Lecture slides, 2007
- Qusay H. Mahmoud: *Getting Started With the Mobile 3D Graphics API for J2ME*. http://developers.sun.com/mobility/apis/articles/3dgraphics/
- Carol Hamer: *Creating a basic M3G file with Blender*. http://bittyjava.wordpress.com/2007/01/04/creating-a-basic-m3g-file-with-blender/
- David Millet, Arthur Tombs et al.: *Blender 3D: Noob to Pro*. http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro
- Mikael Baros: *3D programming tutorial for mobile devices using M3G (JSR 184)*. http://developer.sonyericsson.com/site/global/techsupport/tipstrickscode/mobilejava3d/p_java3d_tutorial_part1_compliments_redikod.jsp
- Kari Pulli et al.: *Developing Mobile 3D Applications With OpenGL ES and M3G*. Eurographics 2006. http://people.csail.mit.edu/kapu/mobile_3D_course/index.html
- Janne Hellsten: *Building scalable 3D apps - Tips & tricks for developers*. http://www.khronos.org/developers/library/seoul_04_2005/Hybrid_Tips-and-Tricks.ppt
- Stephen Chenney: Computer Game Technology. Lecture slides, 2001. http://www.cs.wisc.edu/graphics/Courses/638-f2001/

That’s it!

Thanks for your attention