SIGGRAPH2005

M3G Overview

Tomi Aarnio

Nokia Research Center

Objectives

SIGGRAPH2005

- Get an idea of the API structure and feature set
- Learn practical tricks not found in the spec

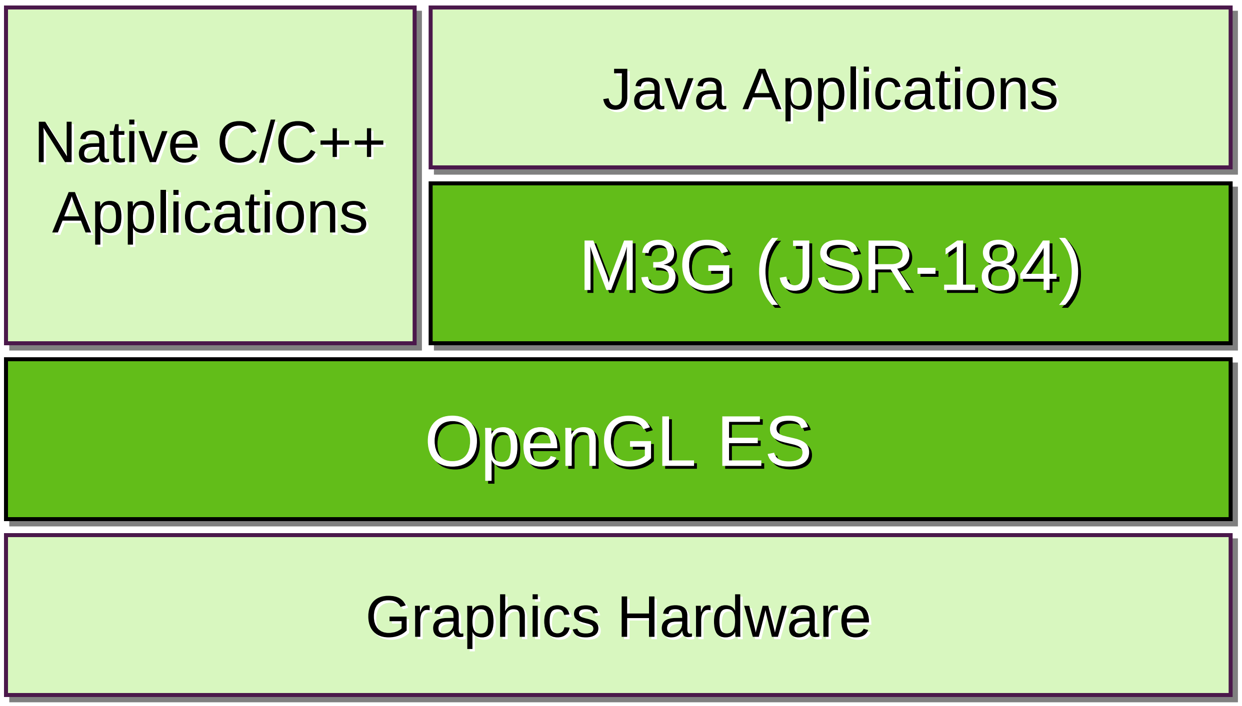
Prerequisites

SIGGRAPH2005

- Fundamentals of 3D graphics
- Some knowledge of OpenGL ES
- Some knowledge of scene graphs

Mobile 3D Graphics APIs

SIGGRAPH2005



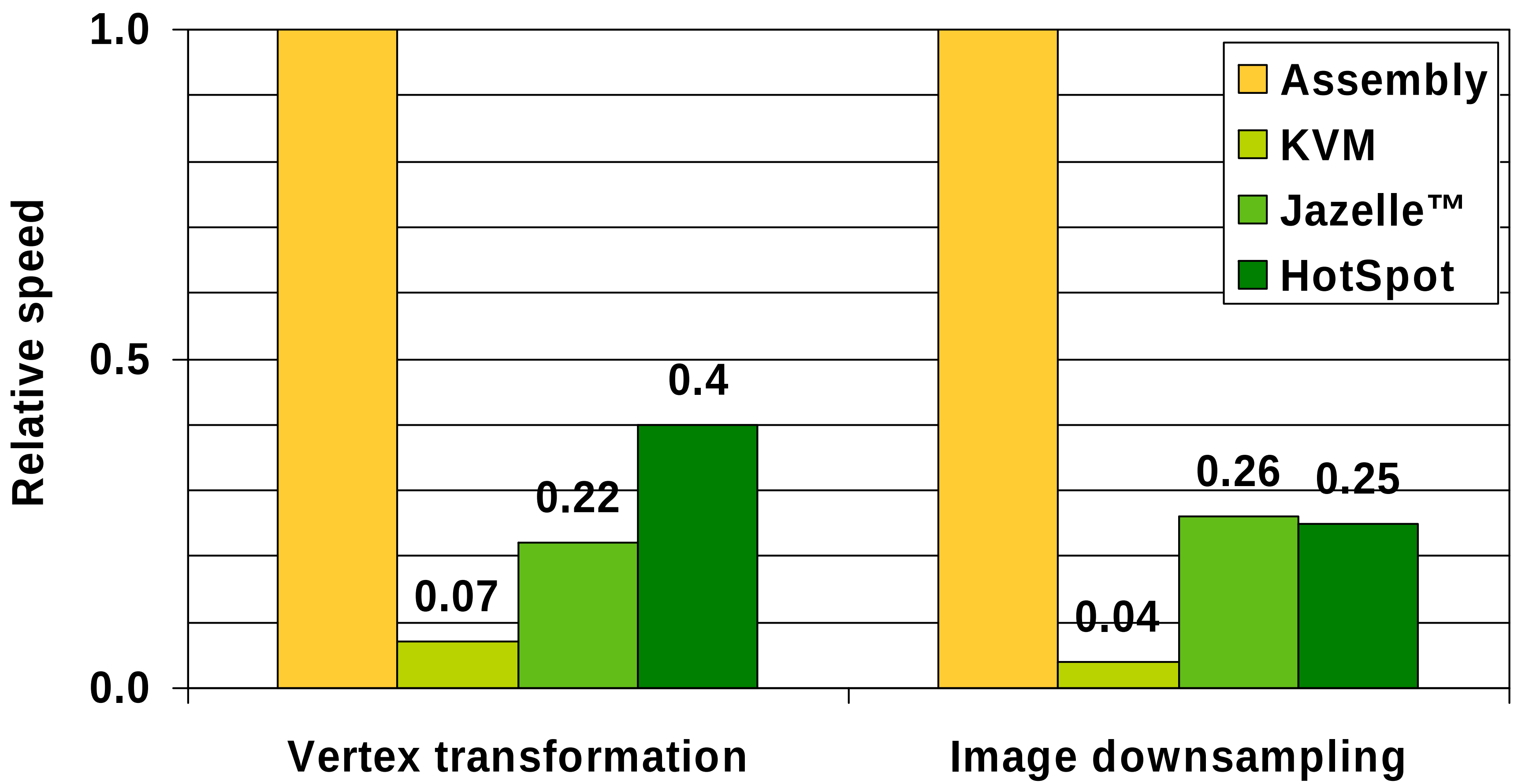
Why Should You Use Java?

SIGGRAPH2005

- It has the largest and fastest-growing installed base
  - 580M Java phones sold by Feb 2005 (source: Sun Microsystems)
  - Nokia alone shipped 125M Java-enabled phones in 2004
  - Less than 12M also supported native Symbian applications
- It increases productivity compared to C/C++
  - Memory protection, type safety ➔ fewer bugs
  - Fewer bugs, object orientation ➔ better productivity

Java Will Remain Slower

SIGGRAPH2005



Benchmarked on an ARM926EJ-S processor with hand-optimized Java and assembly code

Why?

SIGGRAPH2005

- Array bounds & type checking
- Garbage collection
- Expensive Java-to-native calls
- No access to CPU internals
- Stack-based virtual machine
- Unpredictable HotSpot compilers

No Java compiler or accelerator can fully resolve these issues

M3G Overview

SIGGRAPH2005

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

M3G Design Principles

SIGGRAPH2005

#1 **No Java code along critical paths**

- Move all graphics processing to native code
  - Not only rasterization and transformations
  - Also morphing, skinning, and keyframe animation
  - Keep all data on the native side to avoid Java-native traffic

M3G Design Principles

SIGGRAPH2005

#2 | **Cater for both software and hardware**

- Do not add features that are too heavy for software engines
  - Such as per-pixel mipmapping or floating-point vertices
- Do not add features that break the OpenGL 1.x pipeline
  - Such as hardcoded transparency shaders

M3G Design Principles

SIGGRAPH2005

#3 **Maximize developer productivity**

- Address content creation and tool chain issues
  - Export art assets into a compressed file (.m3g)
  - Load and manipulate the content at run time
  - Need scene graph and animation support for that
- Minimize the amount of “boilerplate code”

M3G Design Principles

SIGGRAPH2005

#4	Minimize engine complexity
#5	Minimize fragmentation
#6	Plan for future expansion

Why a New Standard?

SIGGRAPH2005

- OpenGL ES is too low-level
  - Lots of Java code, function calls needed for simple things
  - No support for animation and scene management
  - Fails on Design Principles 1 (performance) and 3 (productivity)
  - …but becomes more practical as Java performance increases
- Java 3D is too bloated
  - A hundred times larger (!) than M3G
  - Still lacks a file format, skinning, etc.
  - Fails on Design Principles 1, 3, and 4 (code size)

M3G Overview

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

The Programming Model

SIGGRAPH2005

- Not an “extensible scene graph”
  - Rather a black box – much like OpenGL
  - No interfaces, events, or render callbacks
  - No threads; all methods return only when done
- Scene update is decoupled from rendering
  - **`render`** → Draws an object or scene, no side-effects
  - **`animate`** → Updates an object or scene to the given time
  - **`align`** → Aligns scene graph nodes to others

Key Classes

Graphics3D

3D graphics context
Performs all rendering

Loader

Loads individual objects and entire scene graphs (.m3g and .png files)

World

Scene graph root node

Rendering State

SIGGRAPH2005

- Graphics3D contains global state
  - Frame buffer, depth buffer
  - Viewport, depth range
  - Rendering quality hints
- Most rendering state is in the scene graph
  - Vertex buffers, textures, matrices, materials, …
  - Packaged into Java objects, referenced by meshes
  - Minimizes Java-native data traffic, enables caching

Graphics3D: How To Use

SIGGRAPH2005

- Bind a target to it, render, release the target

```
void paint(Graphics g) {
   myGraphics3D.bindTarget(g);
   myGraphics3D.render(world);
   myGraphics3D.releaseTarget();
}
```

- Tip: Do not mix 2D and 3D rendering

M3G Overview

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

Renderable Objects

SIGGRAPH2005

Sprite3D

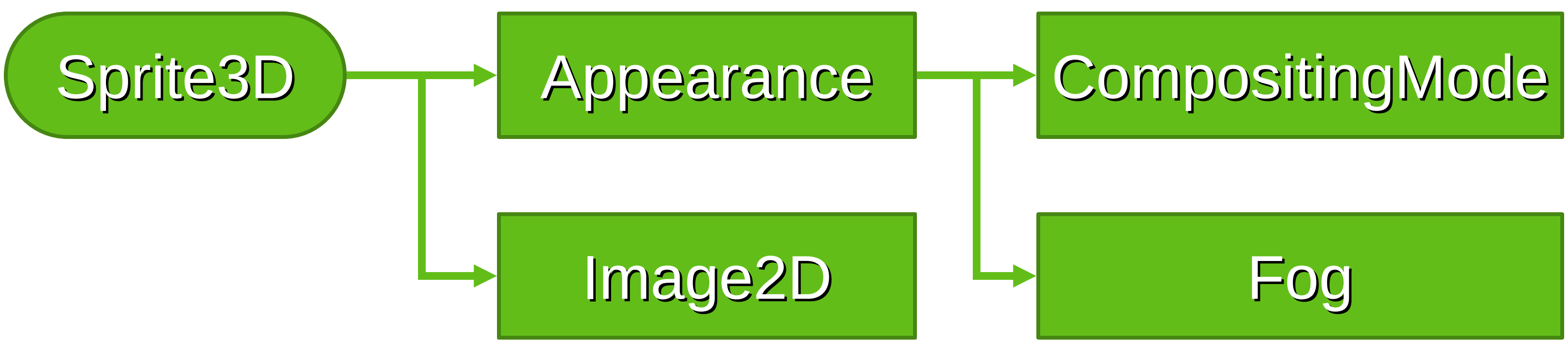
- *2D image placed in 3D space*
- *Always facing the camera*

Mesh

- *Made of triangles*
- *Base class for meshes*

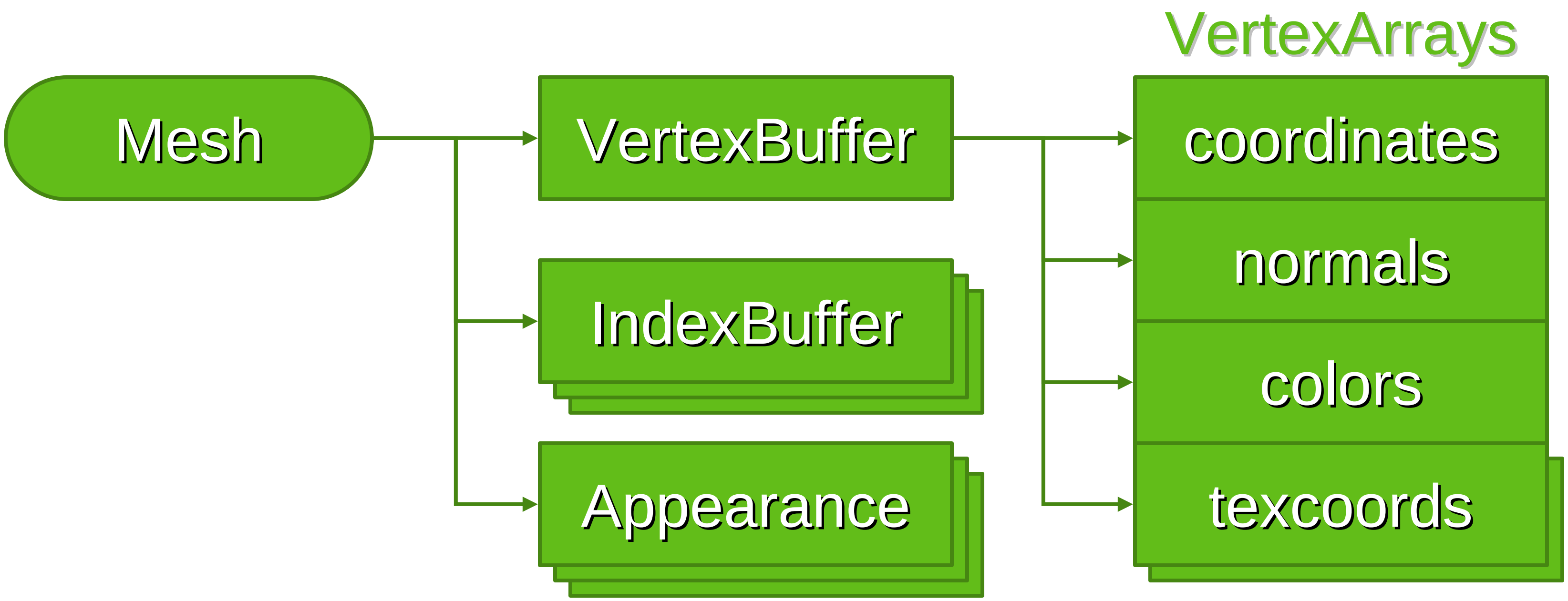
Sprite3D

- 2D image with a position in 3D space
- Scaled mode for billboards, trees, etc.
- Unscaled mode for text labels, icons, etc.



Mesh

- A common VertexBuffer, referencing VertexArrays
- IndexBuffers (submeshes) and Appearances match 1:1



VertexBuffer Types

SIGGRAPH2005

	8-bit	16-bit	32-bit	Float	2D	3D	4D
Vertices	✓	✓	✗	✗	✗	✓	✗
Texcoords	✓	✓	✗	✗	✓	✓	✗
Normals	✓	✓	✗	✗		✓	
Colors	✓		✗	✗		✓	✓

Relative to OpenGL ES 1.1

IndexBuffer Types

SIGGRAPH2005

	8-bit	16-bit	implicit	Strip	Fan	List
Triangles	✗	✓	✓	✓	✗	✗
Lines	✗	✗	✗	✗	✗	✗
Points	✗	✗	✗			✗
Point sprites	✗	✗	✗			✗

Relative to OpenGL ES 1.1 + point sprite extension

Buffer Objects

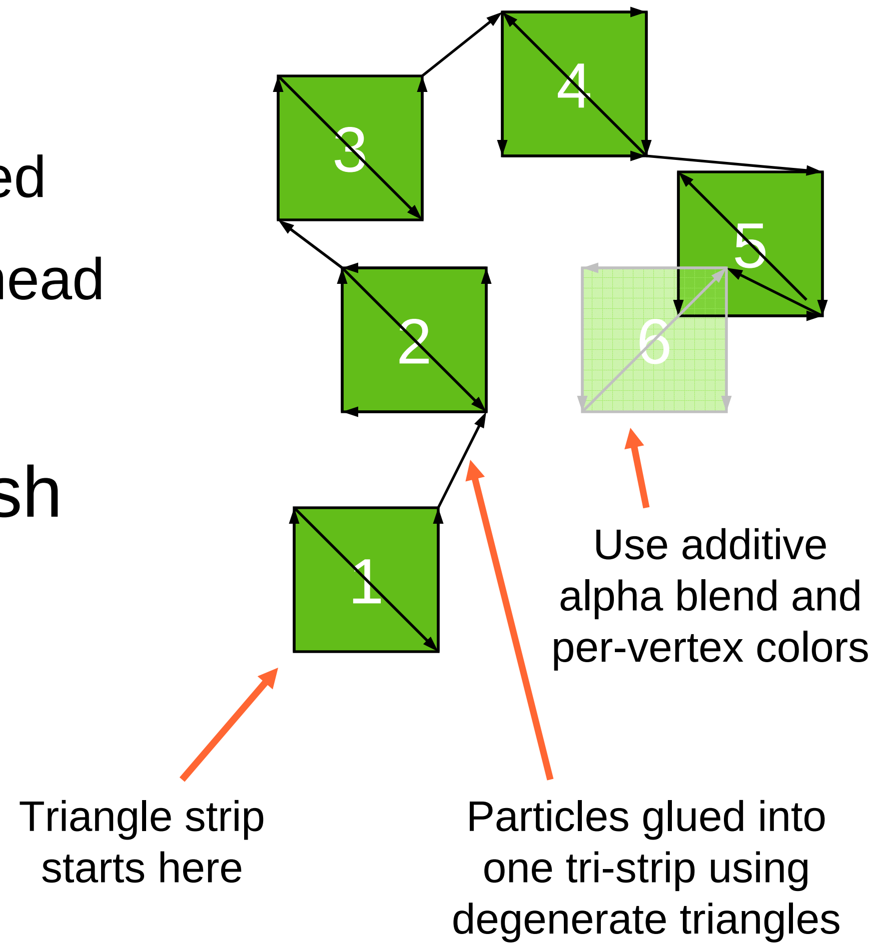
SIGGRAPH2005

- Vertices and indices are stored on server side
  - Very similar to OpenGL Buffer Objects
  - Allows caching and preprocessing (e.g., bounding volumes)
- Tradeoff – Dynamic updates have some overhead
  - At the minimum, just copying in the Java array contents

Tip: Particle Effects

SIGGRAPH2005

- Several problems
  - Point sprites are not supported
  - Sprite3D has too much overhead
- Put all particles in one Mesh
  - One particle == two triangles
  - All glued into one triangle strip
  - Update vertices to animate
    - XYZ, RGBA, maybe UV

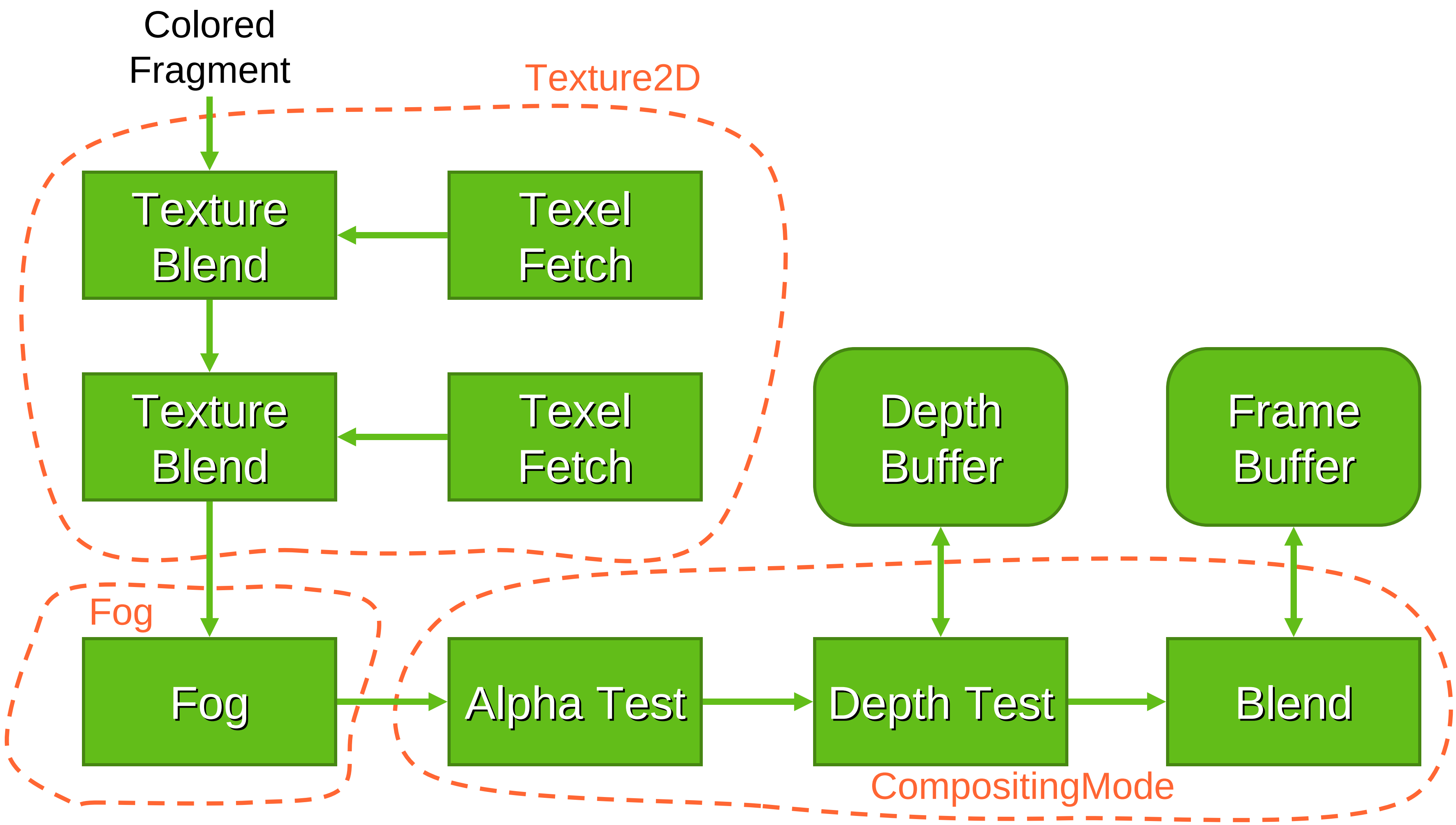


Appearance Components

SIGGRAPH2005

Material	*Material colors for lighting* *Can track per-vertex colors*
CompositingMode	*Blending, depth buffering* *Alpha testing, color masking*
PolygonMode	*Winding, culling, shading* *Perspective correction hint*
Fog	*Fades colors based on distance* *Linear and exponential mode*
Texture2D	*Texture matrix, blending, filtering* *Multitexturing: One Texture2D for each unit*

The Fragment Pipeline



Rendering Tips

SIGGRAPH2005

- Most OpenGL ES performance tips apply
  - Use mipmapping to save in memory bandwidth
  - Use multitexturing to save in T&L and triangle setup
  - SW: Minimize per-pixel operations
  - HW: Minimize shading state changes
- Some of the tips are used by M3G engines
  - Rendering state sorting
  - View frustum culling

Rendering Tips

SIGGRAPH2005

- Use layers to impose rendering order
  - Appearance contains a layer index (integer)
  - Defines a global ordering for submeshes & sprites
  - Useful for multipass rendering, background geometry, etc.
- Use the perspective correction hint – but wisely
  - Usually much faster than increasing triangle count
  - Nokia: 2% fixed overhead, 20% in the worst case
  - Use the hint where necessary, and nowhere else

M3G Overview

SIGGRAPH2005

Design principles

Getting started

Low-level features

The scene graph

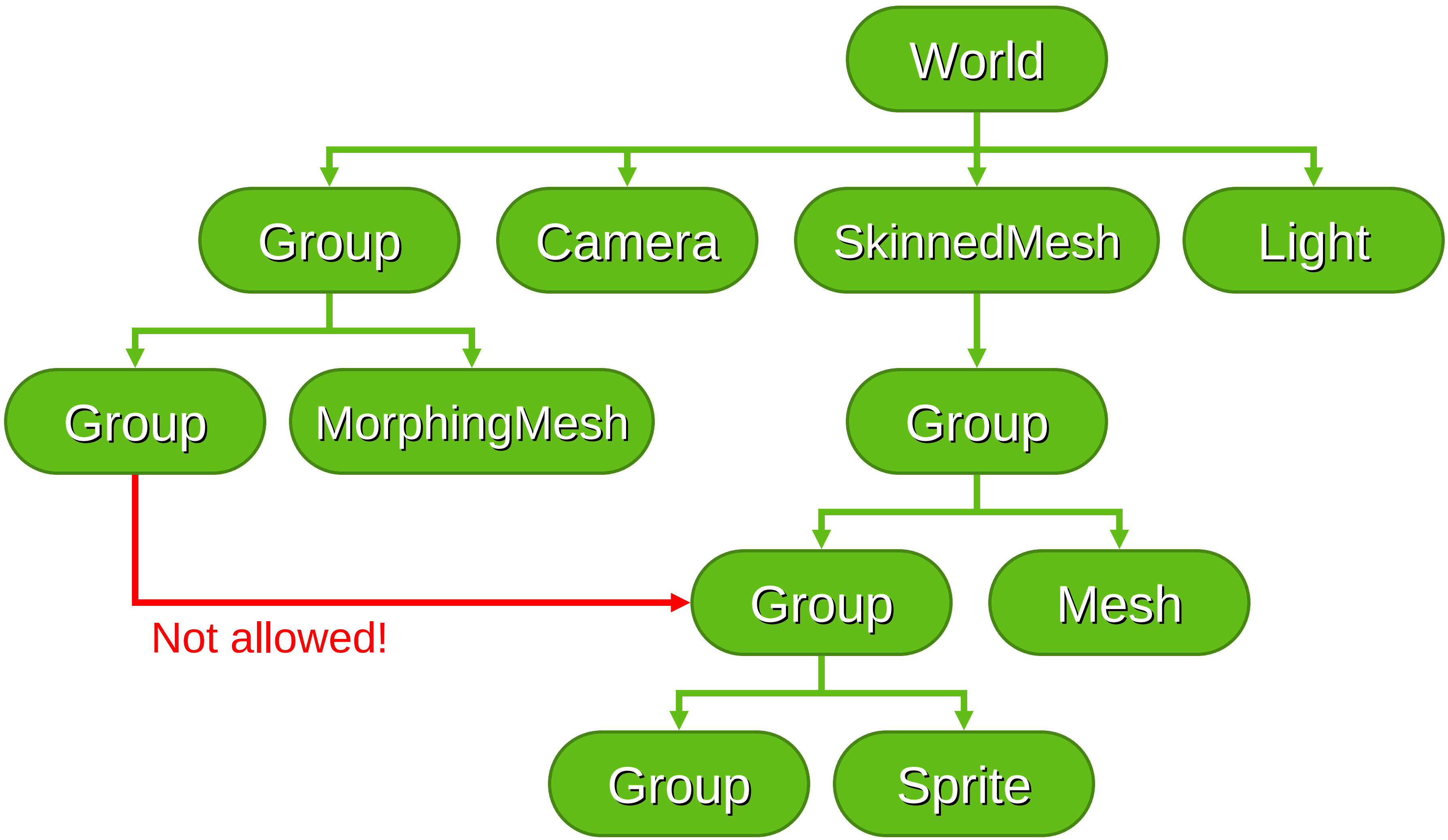
Deforming meshes

Keyframe animation

Summary & demos

The Scene Graph

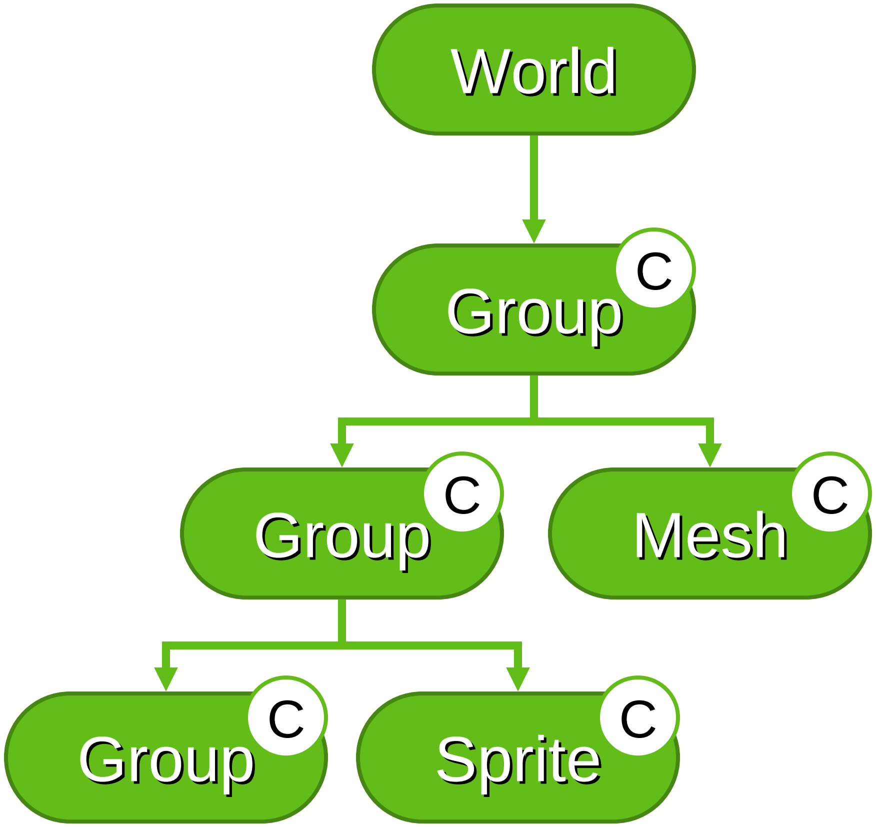
SIGGRAPH2005



Node Transformations

SIGGRAPH2005

- From this node to the parent node
- Composed of four parts
  - Translation T
  - Orientation R
  - Non-uniform scale S
  - Generic 3x4 matrix M
- Composite: **C = T R S M**



Node Transformations

SIGGRAPH2005

Tip: Keep the transformations simple

- Favor the T R S components over M
- Avoid non-uniform scales in S

Tip: Rotating about an arbitrary point (pivot)

- No direct support for pivot translation: $C = T\,P^{-1}\,R\,P\,S\,M$
- Method 1: Combine $T' = T\,P^{-1}$ and $M' = P\,S\,M \rightarrow C = T'\,R\,M'$
  - Drawback: Does not allow S to be animated
- Method 2: Use extra Group nodes

Terrain Rendering

SIGGRAPH2005

Tip: Easy terrain rendering

- Split the terrain into tiles (Meshes)
- Put the meshes into a scene graph
- The engine will do view frustum culling

Tip: Terrain rendering with LOD

- Preprocess the terrain into a quadtree
- Quadtree leaf node == Mesh object
- Quadtree inner node == Group object
- Enable nodes yourself, based on the view frustum

The File Format

SIGGRAPH2005

- Characteristics
  - Individual objects, entire scene graphs, anything in between
  - Object types match 1:1 with those in the API
  - Optional ZLIB compression of selected sections
  - Can be decoded in one pass – no forward references
  - Can reference external files or URIs (e.g. textures)
  - Strong error checking

M3G Overview

SIGGRAPH2005

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

Deforming Meshes

SIGGRAPH2005

MorphingMesh — *Vertex morphing mesh*

SkinnedMesh — *Skeletally animated mesh*

MorphingMesh

SIGGRAPH2005

- Traditional vertex morphing animation
  - Can morph any vertex attribute(s)
  - A base mesh **B** and any number of morph targets $\mathbf{T}_i$
  - Result = weighted sum of morph deltas

$$\mathbf{R} = \mathbf{B} + \sum_i w_i \left(\mathbf{T}_i - \mathbf{B} \right)$$

- Change the weights w_i to animate

MorphingMesh

Base

Target 1
eyes closed

Target 2
mouth closed

Animate eyes
and mouth
independently

SkinnedMesh

SIGGRAPH2005

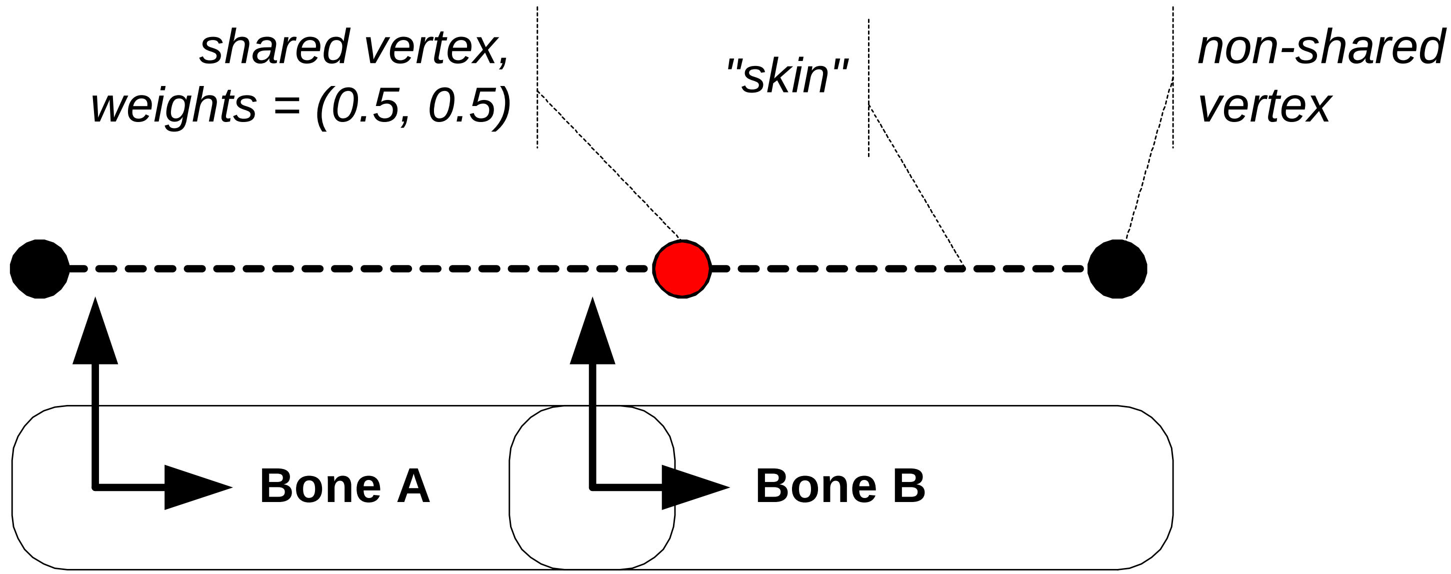
- Articulated characters without cracks at joints
- Stretch a mesh over a hierarchic “skeleton”
  - The skeleton consists of scene graph nodes
  - Each node (“bone”) defines a transformation
  - Each vertex is linked to one or more bones

$$v' = \sum_i w_i \mathbf{M}_i \mathbf{B}_i v$$

  - $\mathbf{M}_i$ are the node transforms – v, w, $\mathbf{B}$ are constant

SkinnedMesh

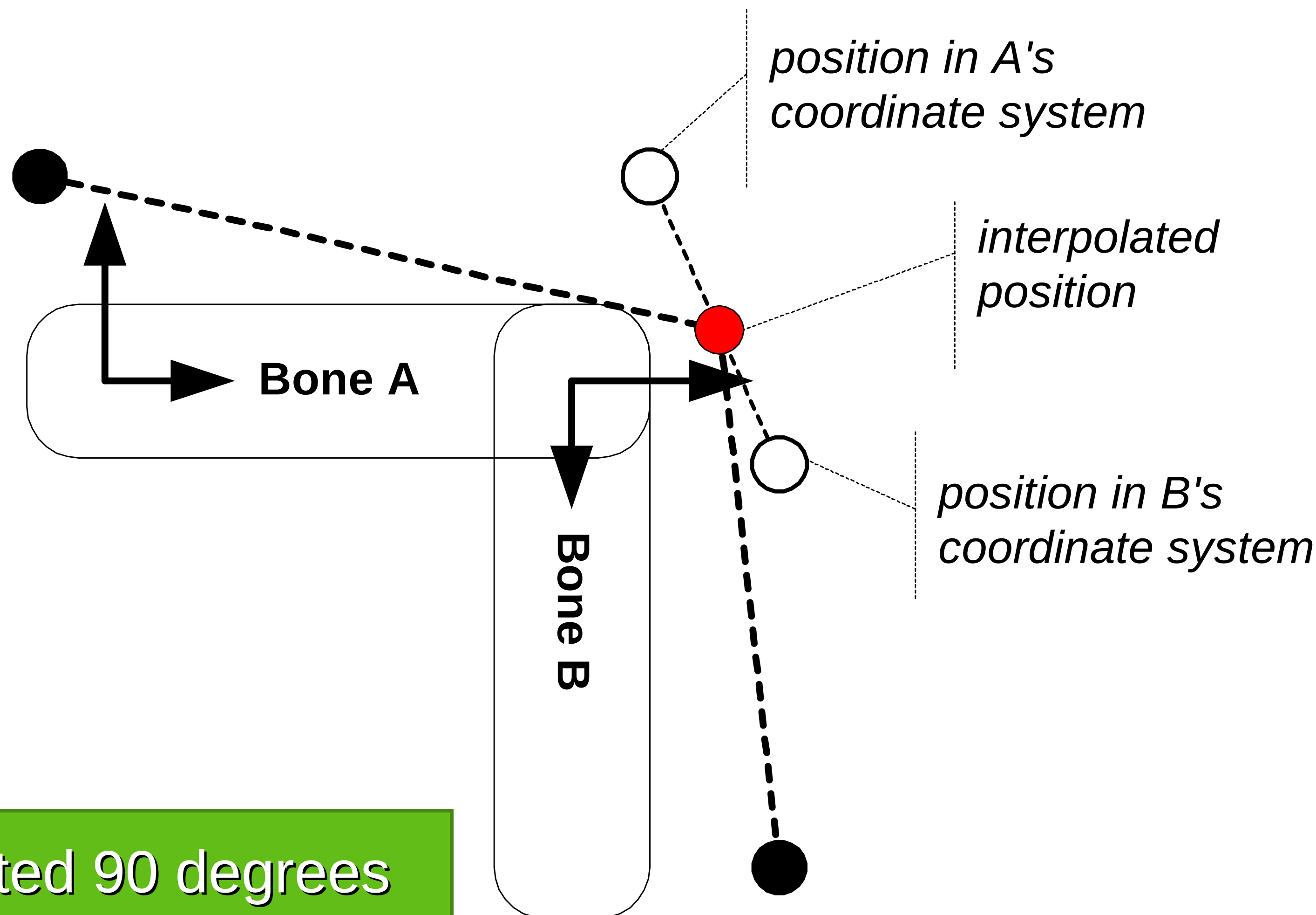
SIGGRAPH2005



Neutral pose, bones at rest

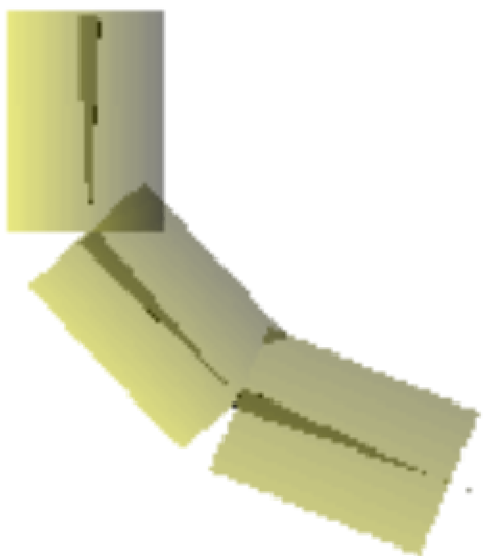
SkinnedMesh

SIGGRAPH2005

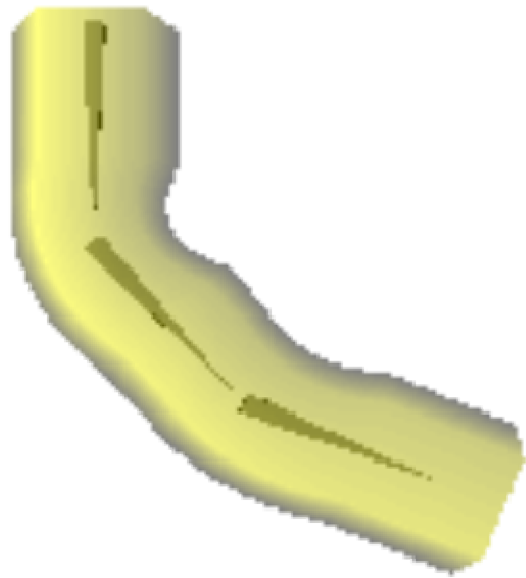


Bone B rotated 90 degrees

SkinnedMesh



No skinning



Smooth skinning
two bones per vertex

M3G Overview

SIGGRAPH2005

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

Animation Classes

SIGGRAPH2005

KeyframeSequence

Storage for keyframes
Defines interpolation mode

AnimationController

Controls the playback of one or more sequences

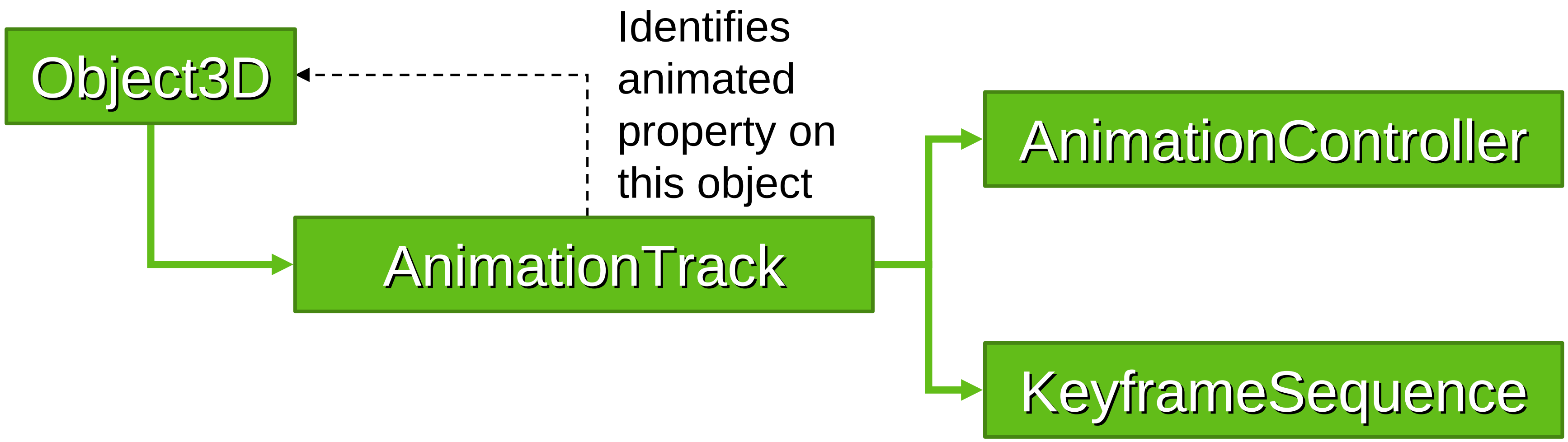
AnimationTrack

A link between sequence, controller and target

Object3D

Base class for all objects that can be animated

Animation Classes



KeyframeSequence

KeyframeSequence

Keyframe is a time and the value of a property at that time

Can store any number of keyframes

Several keyframe interpolation modes

Can be open or closed (looping)

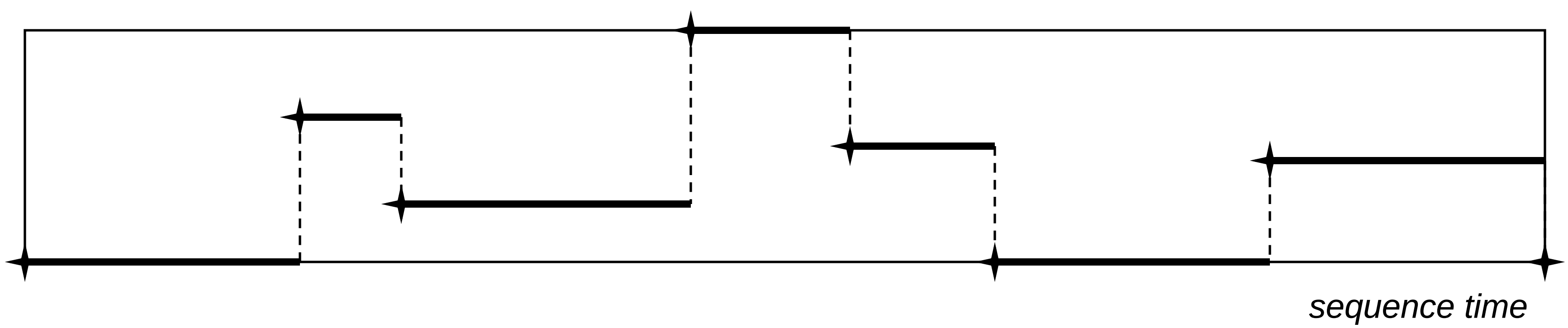


Diagram courtesy of Sean Ellis, Superscape

KeyframeSequence

SIGGRAPH2005

KeyframeSequence

Keyframe is a time and the value of a property at that time

Can store any number of keyframes

Several keyframe interpolation modes

Can be open or closed (looping)

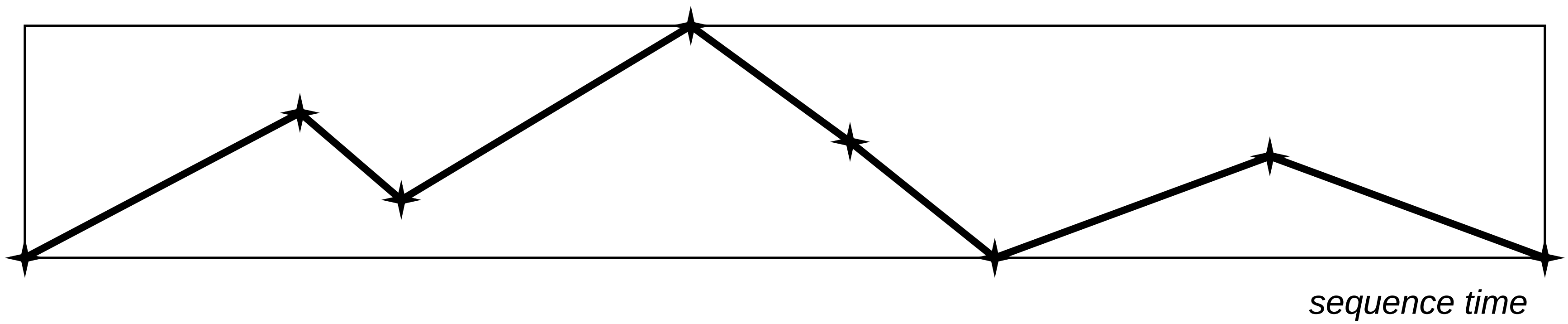


Diagram courtesy of Sean Ellis, Superscape

KeyframeSequence

SIGGRAPH2005

KeyframeSequence

Keyframe is a time and the value of a property at that time

Can store any number of keyframes

Several keyframe interpolation modes

Can be open or closed (looping)

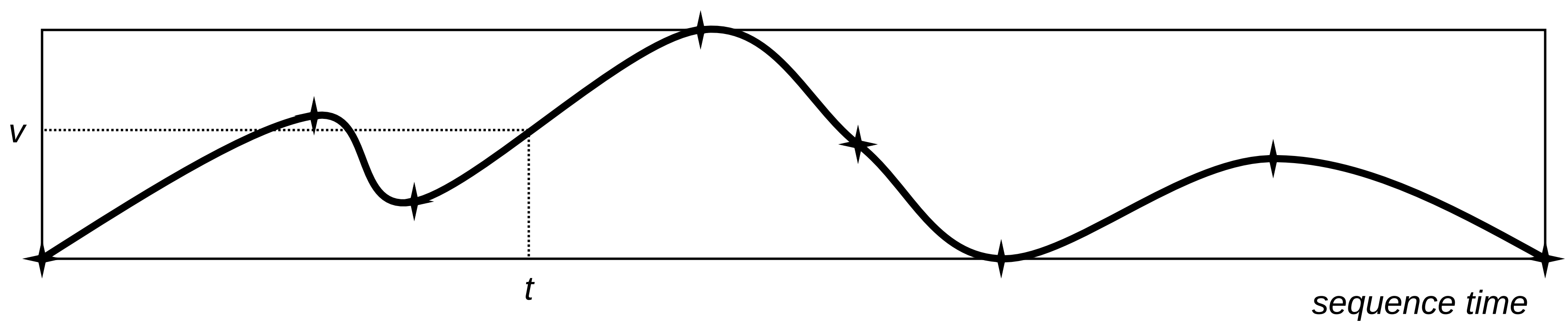


Diagram courtesy of Sean Ellis, Superscape

AnimationController

AnimationController

Can control several animation sequences together

Defines a linear mapping from world time to sequence time

Multiple controllers can target the same property

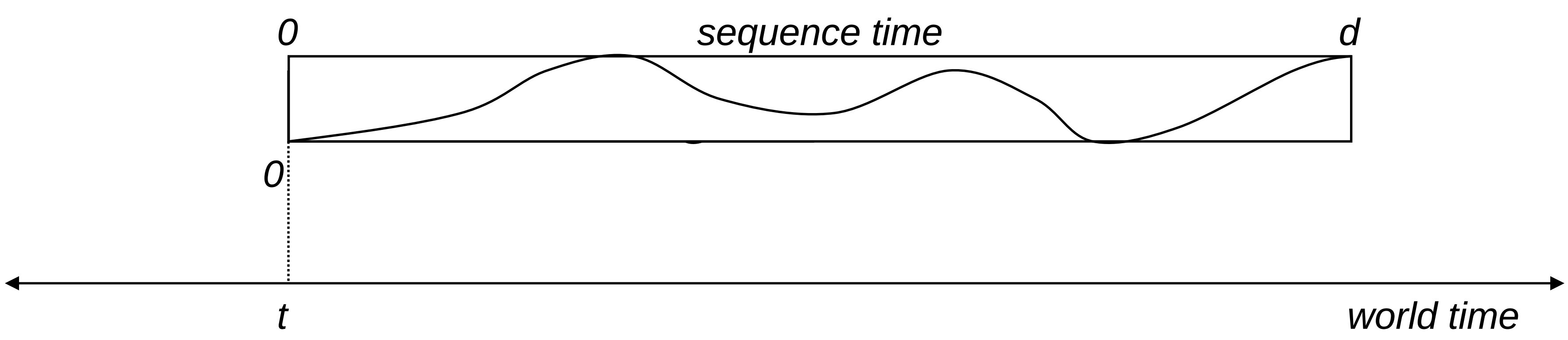


Diagram courtesy of Sean Ellis, Superscape

Animation

SIGGRAPH2005

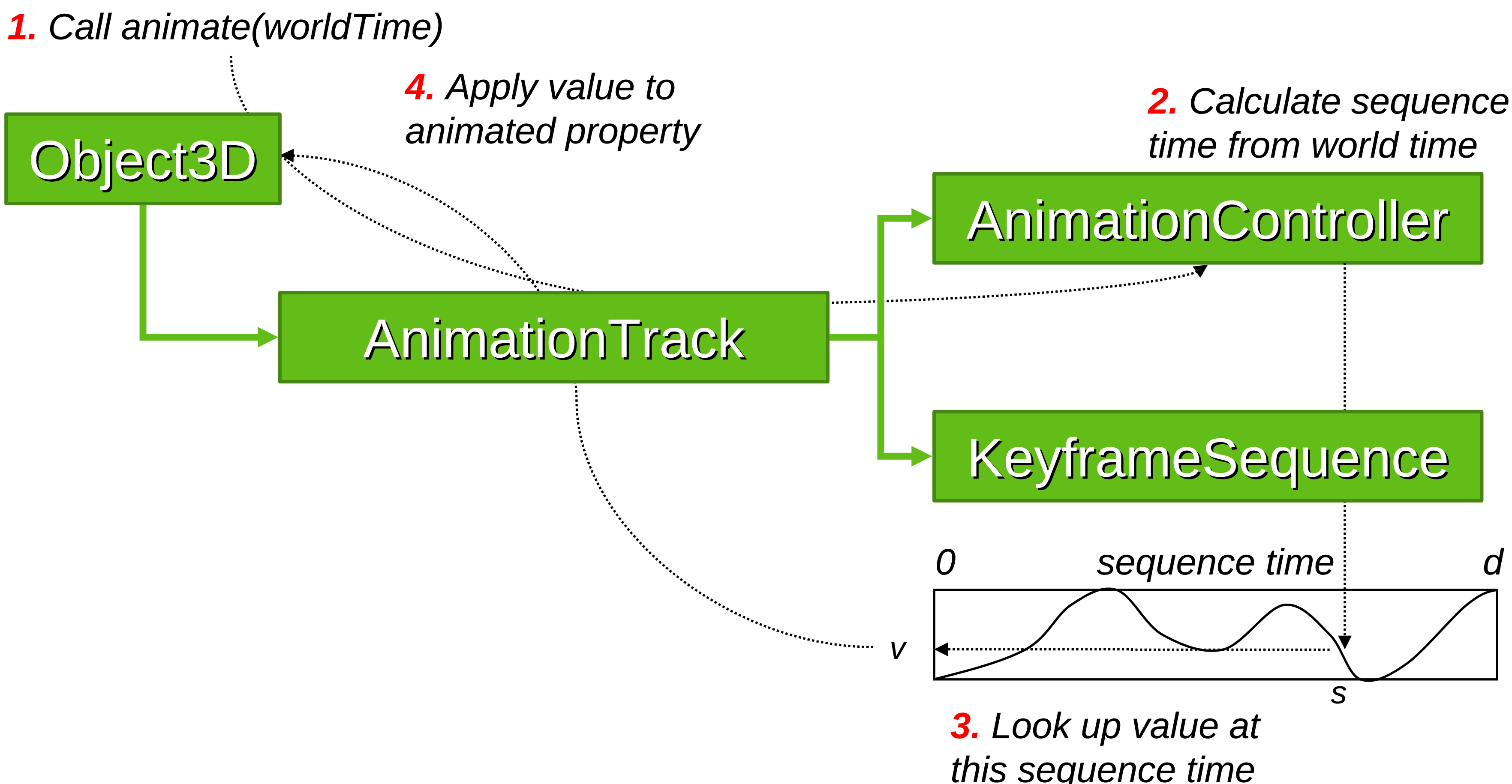


Diagram courtesy of Sean Ellis, Superscape

Animation

SIGGRAPH2005

Tip: You can read back the animated values

- Much faster than Java if you need floating point interpolation
- Target N-dimensional tracks (N > 4) to a dummy MorphingMesh

Tip: Interpolate quaternions as ordinary 4-vectors

- SLERP and SQUAD are slower, but need less keyframes
- Quaternions are automatically normalized before use

M3G Overview

Design principles

Getting started

Low-level features

The scene graph

Deforming meshes

Keyframe animation

Summary & demos

Predictions

SIGGRAPH2005

- Resolutions will grow rapidly from 128x128 to VGA
  - Drives graphics hardware into all high-resolution devices
  - Software rasterizers can't compete above 128x128
- Bottlenecks will shift to Physics and AI
  - Bottlenecks today: Rasterization and <u>any</u> Java code
  - Graphics hardware will take care of geometry and rasterization
  - Java hardware will increase performance to within 50% of C/C++
- Java will reinforce its position as the dominant platform

Summary

SIGGRAPH2005

- M3G enables real-time 3D on mobile Java
  - By minimizing the amount of Java code along critical paths
  - Designed for both software and hardware implementations
- Flexible design leaves the developer in control
  - Subset of OpenGL ES features at the foundation
  - Animation & scene graph features layered on top

Installed base growing by the millions each month

Demos

Q&A

Thanks: Sean Ellis, Kimmo Roimela,
Nokia M3G team, JSR-184 Expert Group

SIGGRAPH2005