
# Mobile graphics API Overview


Michael Doggett

Department of Computer Science

Lund University

# Register

- Please check to see if your name is on the list, if not add it to the end.

# First part of this course:

- Is about writing a ***mobile 3D application*** that *works* on your (modern) mobile phone ***today***
- Programming assignment 1: simple game
- To pull that off, we need ***graphics APIs***

# Important features of graphics APIs (for mobiles)?

- Open for everyone
- Cross-platform API
- Should work everywhere
  - Write application only once
- Should be “neat”
  - “Lean & mean” – no unnecessary fat
  - Powerful
  - Expressive
  - Etc

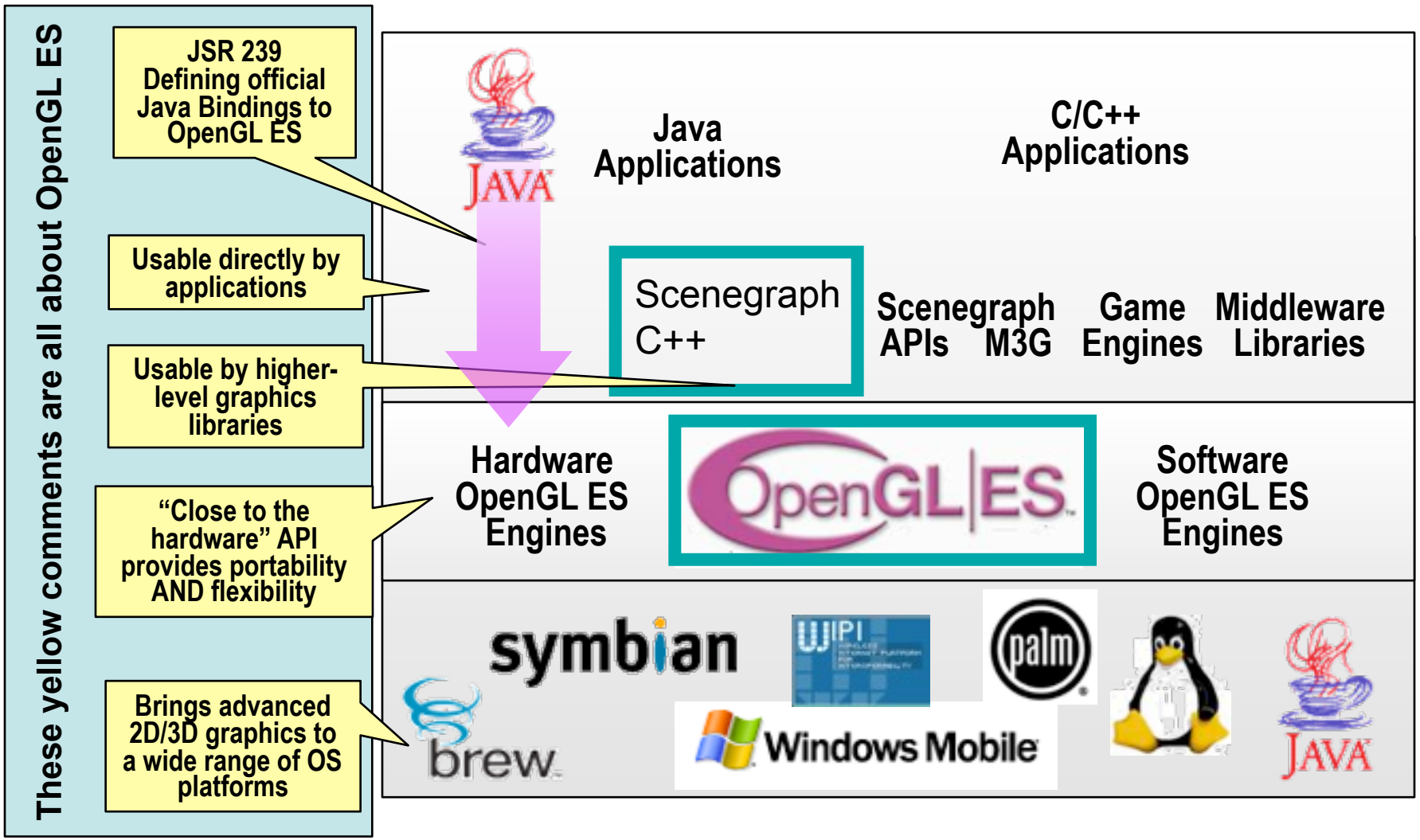
# Mobile APIs

**(there are other APIs as well – more about that later today)**


**ES=Embedded systems**

- Low-level API: focus on rendering triangles
  - Under the hood: either software implementation or dedicated graphics hardware
- Built upon OpenGL for desktop computers
  - Highly successful real-time graphics API
- There is Java API for this (JSR239)

# Different layers




[Slide courtesy of Tom Olsen, Texas Instruments]


**...involves many companies....**
**[i.e., not only one, or a few]**

# OpenGL ES

- Is for everyone!
  - Example:
    - PS3 uses it
    - IPhone uses it

**PlayStation III**

**IPhone**

SGX 535

- Many mobile phones use it today:

**NOKIA N900**

PowerVR SGX 530

**Sony Ericsson Satio**

OpenGL ES 2 through PowerVR SGX 530


**MS Zune**

Nvidia Tegra

**Sony Ericsson W900i**

**GoForce 4800**

# OpenGL ES differences

- OpenGL 1.0 released in 1992 for desktops
- OpenGL too big for mobiles
  - Mobile has limited resources (e.g., memory, floating-point hw, etc)
- ES is mostly a subset of full OpenGL
  - Eliminated redundant/unneeded/expensive functionality
  - Compact and efficient
    - <=50 kB footprint possible
- Let's look at some design decisions...

# Example OpenGL / OpenGL ES 1 differences

## Functionality: in / out? (2/7)

SIGGRAPH2005

- Remove old complex functionality
  - glBegin – glEnd (**OUT**); vertex arrays (**IN**)
  - new: coordinates can be given as bytes

```
glBegin(GL_POLYGON);
glColor3f (1, 0, 0);
glVertex3f(-.5, -.5, .5);
glVertex3f( .5, .5, .5);
glColor3f (0, 1, 0);
glVertex3f( .5,-.5, .5);
glVertex3f(-.5, .5, .5);
glEnd();
```

```
static const GLbyte verts[4 * 3] =
{    -1,  1,  1,      1,  1,  1,
      1, -1,  1,     -1, -1,  1 };
static const GLubyte colors[4 * 3] =
{   255,  0,  0,    255,  0,  0,
      0,255,  0,      0,255,  0 };
glVertexPointer( 3,GL_BYTE,0, verts );
glColorPointerf( 3,GL_UNSIGNED_BYTE,
                 0, colors );
glDrawArrays( GL_TRIANGLES, 0, 4 );
```

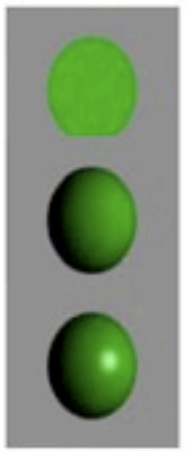
OpenGL|ES

Slide courtesy of
Kari Pulli, Nokia

# Functionality: in / out? (6/7)

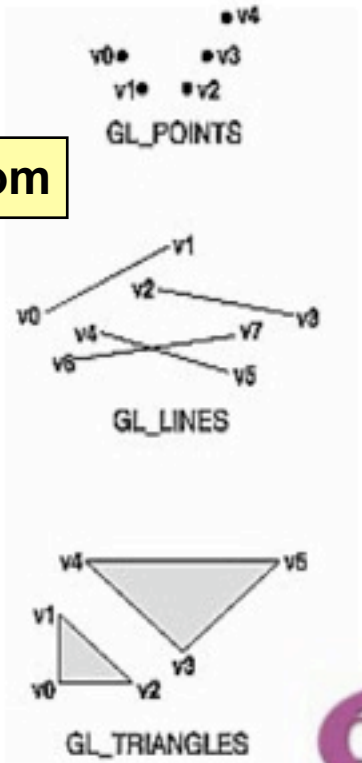
SIGGRAPH2005

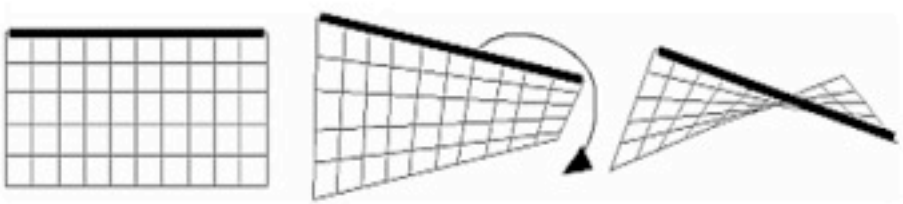
**OUT in GL ES 2.0**



- Almost full OpenGL light model **IN**
  - back materials, local viewer, separate specular **OUT**
- Primitives
  - **IN:** points, lines, triangles
  - **OUT:** polygons and quads

**The rendering atom**



Slide courtesy of
Kari Pulli, Nokia

OpenGL|ES

# More differences...

## Floats vs. fixed-point


- Accommodate both
  - integers / fixed-point numbers for efficiency
  - floats for ease-of-use and being future-proof
- Details
  - 16.16 fixed-point: add a decimal point inside an int

```
glRotatef( 0.5f,    0.f   , 1.f,    0.f    );
                    vs.
glRotatex( 1 << 15,    0   , 1 << 16,    0    );
```

  - get rid of doubles

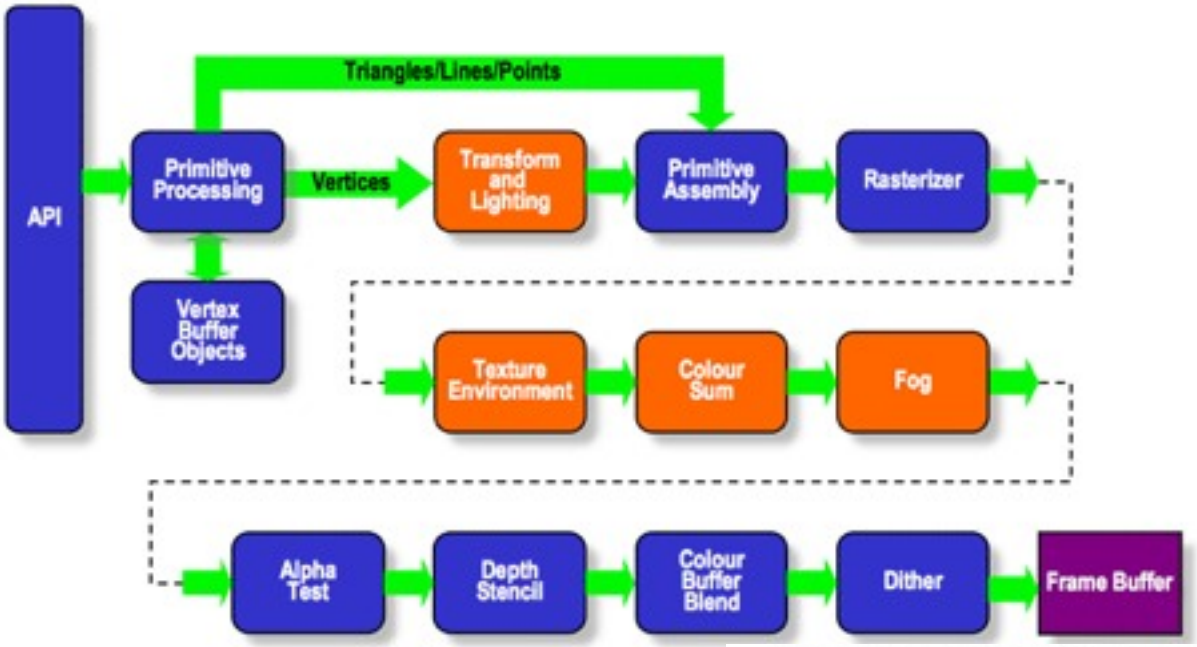
Slide courtesy of
Kari Pulli, Nokia

# The result...

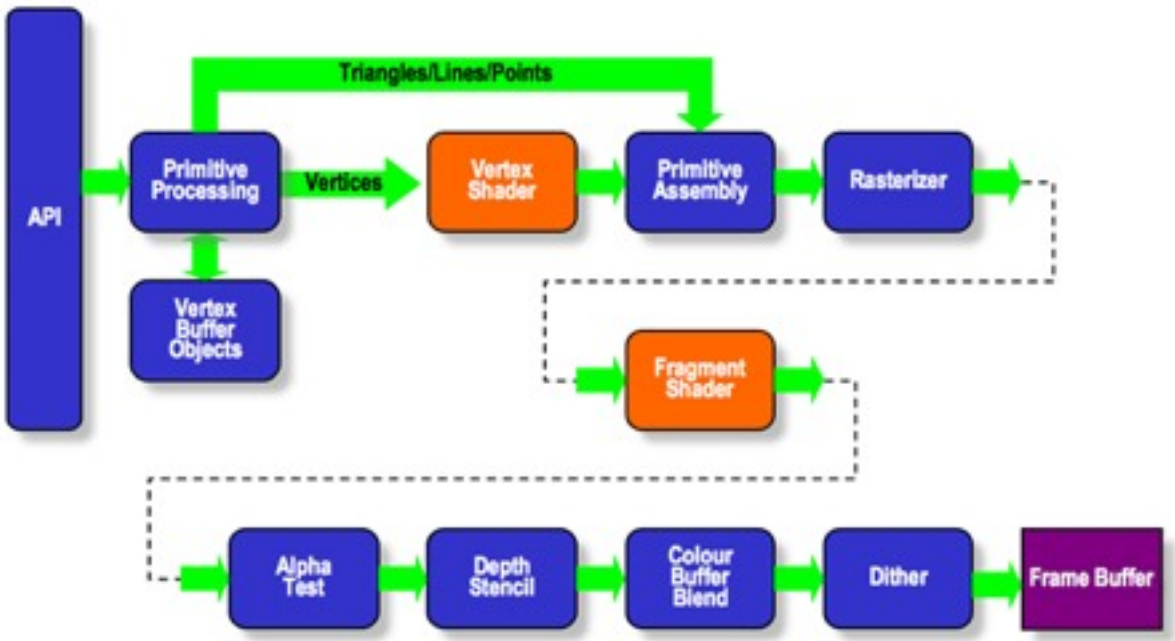
- Compact API
- Support for ”everything” needed for mobile graphics
- Versions:
  - OpenGL ES 1.x is for fixed-function hardware
    - 1.0 in 2003, and 1.1 in 2004
  - OpenGL ES 2.0 is for programmable shader hardware

# 1.x vs 2.x

## OpenGL ES 1.x Fixed Function Pipeline




## OpenGL ES 2.0 Programmable Pipeline



# OpenGL ES 2.0: Shaders Go Mobile

- Graphics industry has going through a programmable revolution
  - Shader programs running on the GPU enable amazing new visual effects
- Graphics APIs will need to support shading languages
  - Enabling new visual effects to be created by developers

**Doom 3's Zombies**

**Unreal's Rocks**

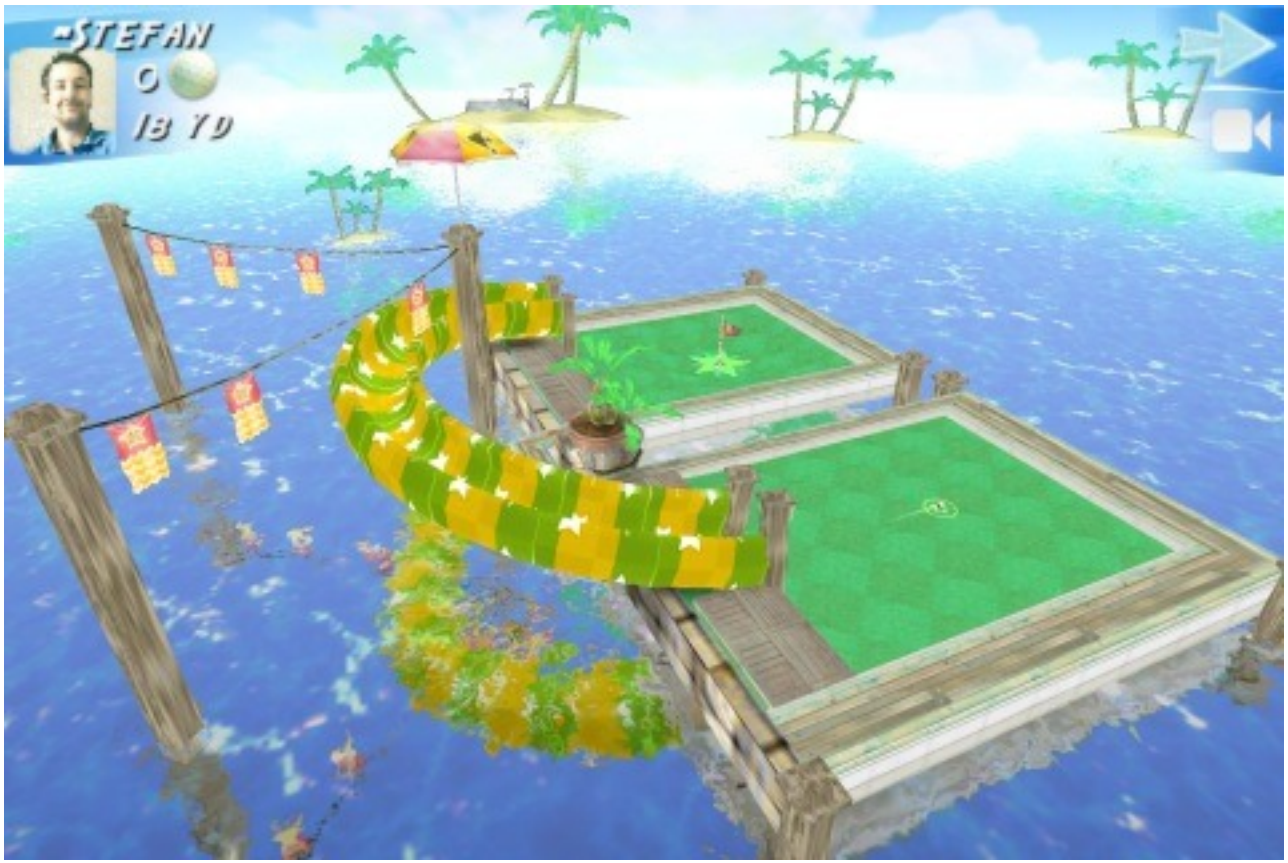
**Halo's Ice**

**Far Cry's Water**

Slide courtesy of
Tom Olson,
Texas Instruments Inc.

# Done with OpenGL ES...

# Done with OpenGL ES 2.0...

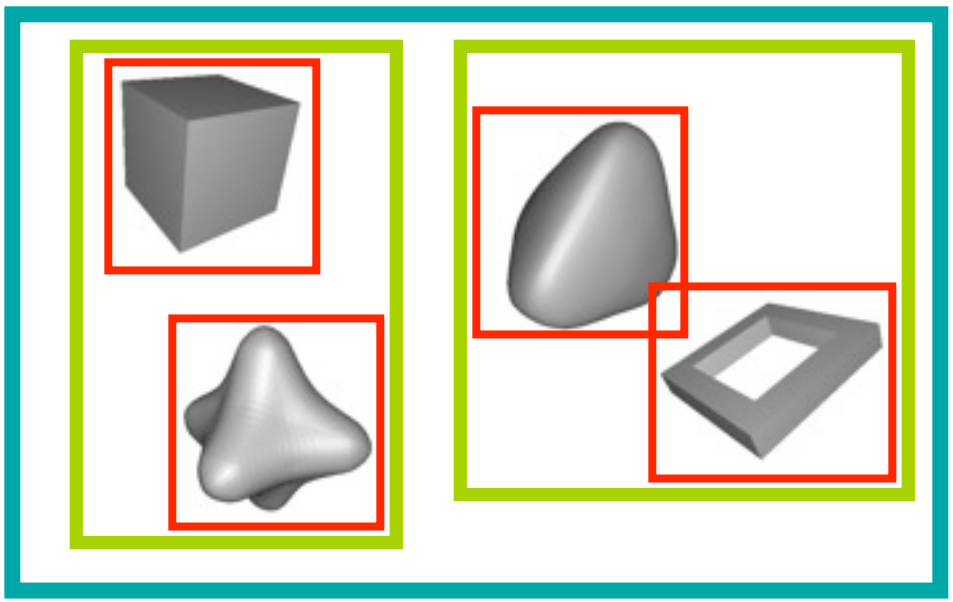



- Adrenaline Golf uses GL2 shaders for water reflection on iPhone 3GS

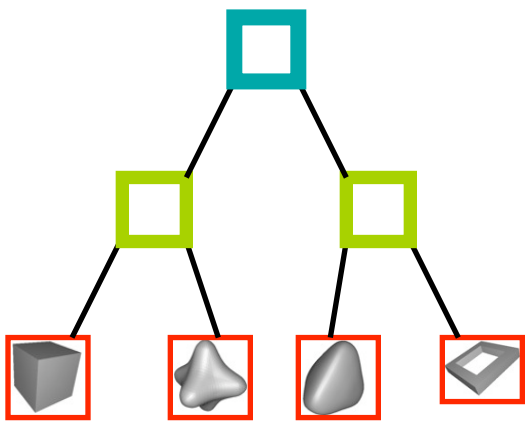
# Scene Graphs

- Organizes geometry and other things in a hierarchy (usually a tree-like structure) called ***the scene graph***

In 2D space



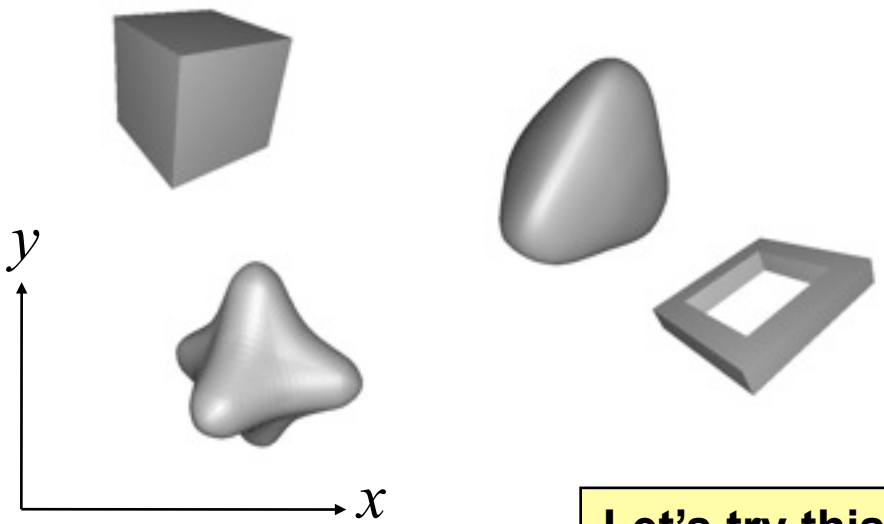
Scene graph

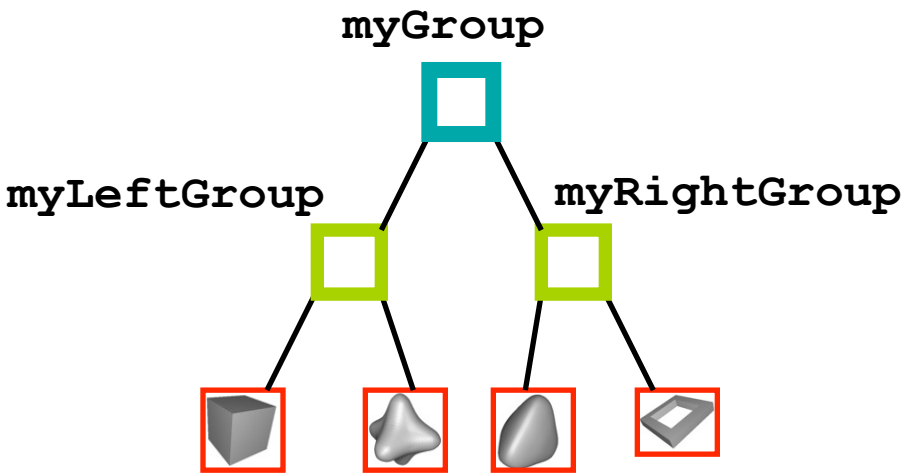


# Scene graph: what for?

- Example: at each node, we can set a transform
  - Gives us hierarchical animation
    - E.g., the planets rotate around the sun, but moons rotate around planets (which rotate around the sun)

**Original positions of objects:**



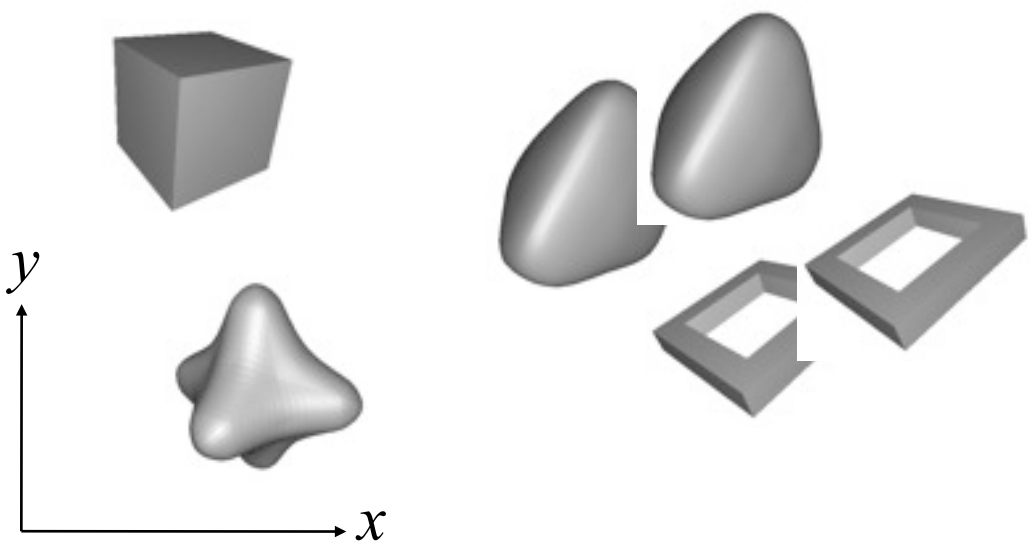



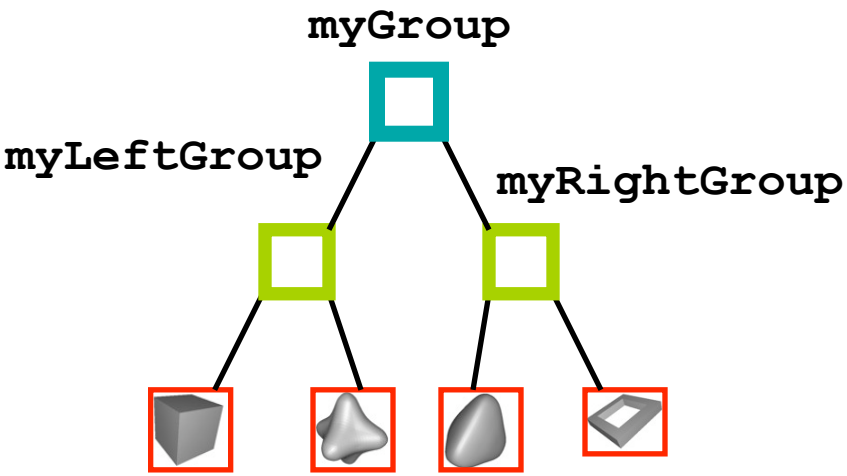

**Let's try this:**

```
myRightGroup.translate(2,1,0);
```

# Scene graph example (cont'd)

```
myRightGroup.translate(2,1,0);
```



# Scene graph example (cont'd)

- Look in the RenderChimp.h for a quick intro and other .h files for comments
- More details in Magnus' SceneGraph intro on Tuesday

```
Group *g;
g =
sceneGraph.createGroup(...);
g->attachChild(Node *child);
g->detachChild(Node *child);
g->getChild(u32 index);
etc
Example: myGroup-
>attachChild(myLeftGroup);
```

```
TriangleMesh *mesh0, *mesh1;
Light *myLight;
Camera *myCamera;
Sprite *mySprite;
mesh0->attachChild(mesh1);
...and more.
```

```
mesh0 =
sceneGraph.createTriangleMesh(...);
mesh0->translate(...);
mesh0->scale(...);
mesh0->rotate(...);
mesh0->setTranslate(...);
mesh0->setScale(...);
etc
```

# There are other APIs as well

- Mobile Direct3D
  - For Microsoft Windows CE and Windows Mobile

# OpenGL Shading Language (GLSL)

- OpenGL **ES** 2.0 shading language
- C like language for programming GPUs
- Basically the same language for vertex and fragment shaders/programs

# GLSL - Types

- Types
  - `float, int, bool,`
    - `vec2, vec3, vec4, etc...`
  - `sampler2D, samplerCube`
- Type qualifiers
  - `const` `// compile-time constant`
  - `uniform` `// constant across primitive`
  - `attribute` `// vertex shader per vertex input`
  - `varying` `// interpolated data from vertex to pixel`
  - Different quantities for vertex and fragment shader
  - Precision qualifiers for float and int
    - lowp (s1.8fixed), mediump (se5m10), highp (se8m16)
    - Must be specified for floats in fragment shader

# GLSL - Built-In Types

- Most GLSL built-ins removed
- Vertex shader outputs
  - `gl_Position`
- Fragment shader outputs
  - `gl_FragColor, gl_FragDepth`

# OpenGL Shading Language

- Structures and arrays
- Operators work with 1-4 component types
  - `{x, y, z, w}, {r, g, b, a}, {s, t, p, q}`
  - `vector.wzyx // swizzles`
- User defined functions
- Built-in functions
  - `sin, cos, pow, log2, sqrt, normalize, noise, …`

# OpenGL Shading Language

- Flow control
  - `if, if-else, for, while, do-while`
  - `discard // fragment only`
- Preprocessor directives
  - e.g `#define, #undef, #if, #else, #endif, etc`
  - comments `//  /* … */`

# Simple shaders

- Vertex shader

```
uniform mat4 ModelViewProjection;
attribute vec3 Vertex;
attribute vec3 Color;
varying mediump vec4 icolor;

void main(void) {
    gl_Position = ModelViewProjection * vec4(Vertex.xyz, 1.0);
   icolor = Color;
}
```

- Fragment shader

```
varying mediump vec4 icolor;

void main() {
    gl_FragColor = icolor;
}
```

# Shader development

- ATI RenderMonkey
- Nvidia FX Composer
- Mac OpenGL Shader Builder

# Next week

- Time to start working on programming assignment #1: Simple game
  - Will be released tomorrow
- Tuesday: seminar/lecture at 10:00
  - About Scene Graph, C++ and the assignment
- Check out the discussion forum every now and then
- Ahhh, and then after that...
  - ...the real fun begins!

# The end