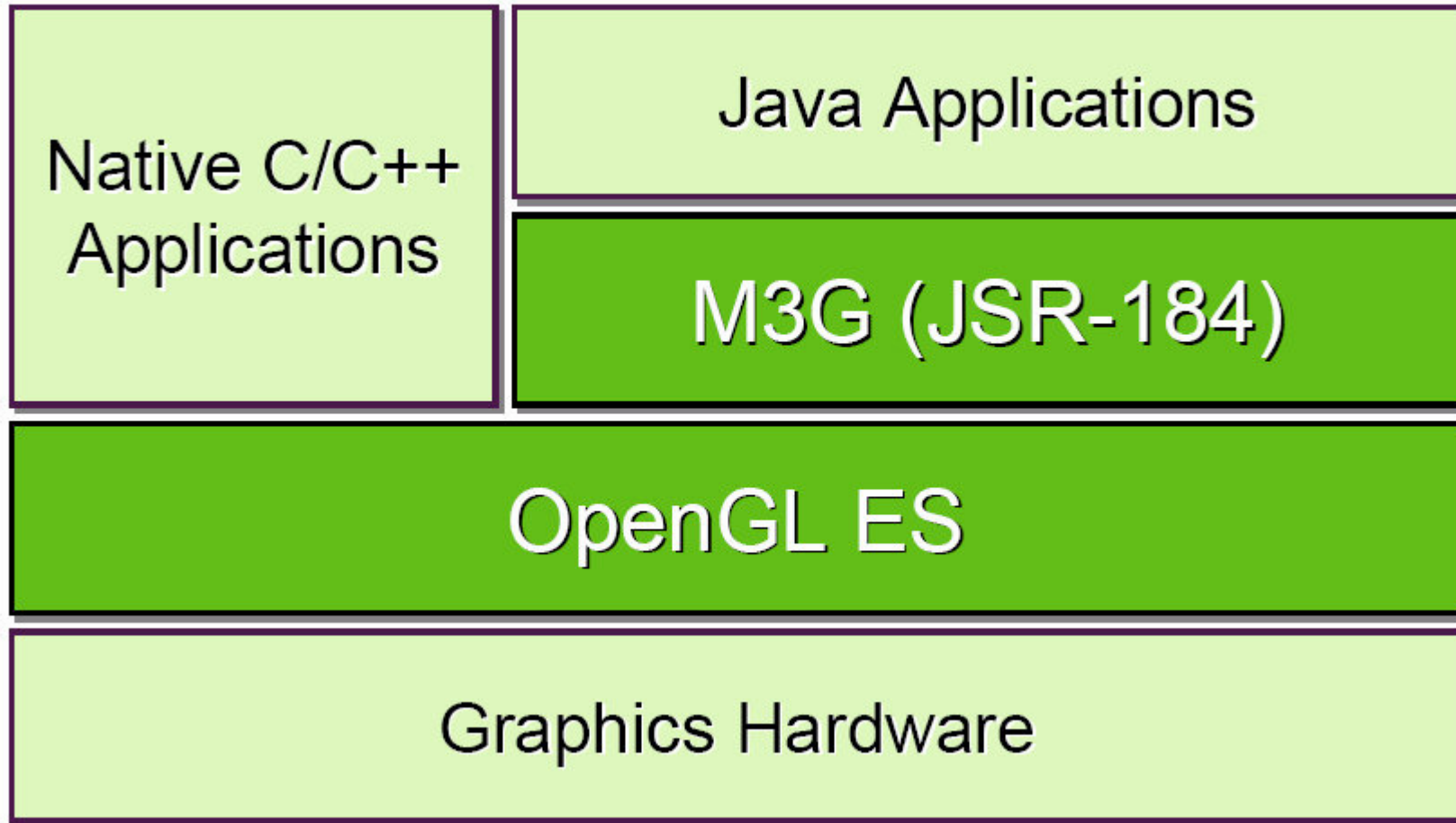


# M3G - overview



# M3G – "Core" classes

Graphics3D

Top level Class

- Rendering commands
- Window related commands
  - Clear screen, Set render area etc.

Loader

Resource loading

- M3G files
  - Objects, Animations, entire scene graph
- PNG files
  - Images (textures)

World

Scene graph root node

# Rendering loop

- Bind the Graphics3D object to a canvas (graphics object), render the world, release the canvas.

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

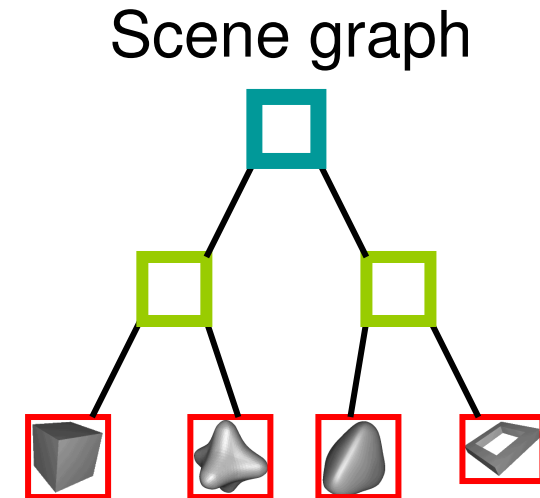
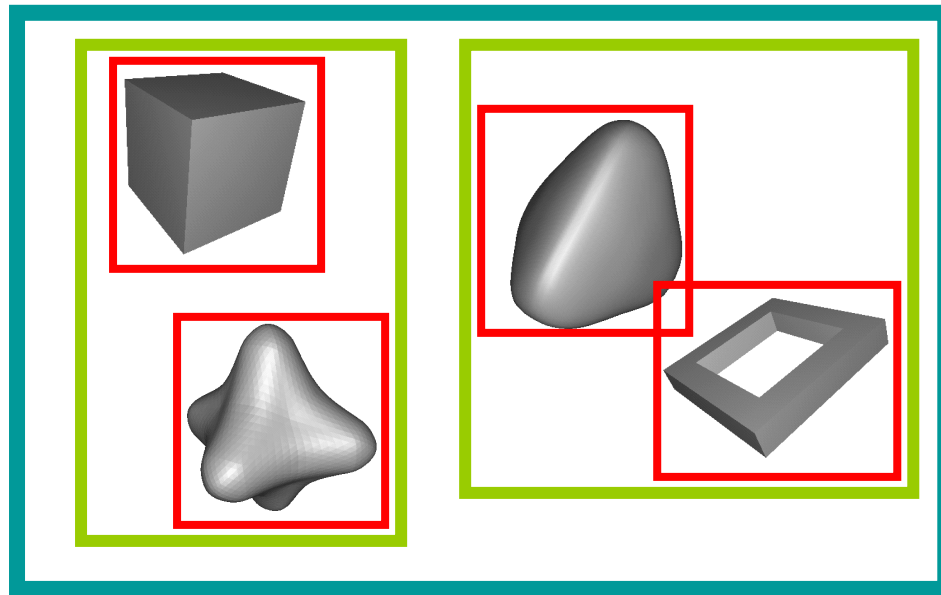
- Can render separate objects or vertex lists, but that is not needed in the assignment

# M3G is scene-graph based...

(very powerful thing)

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

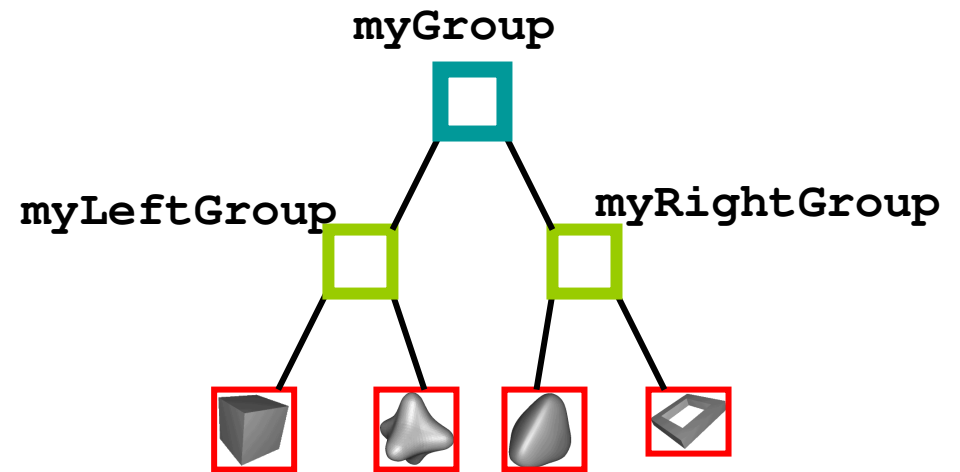
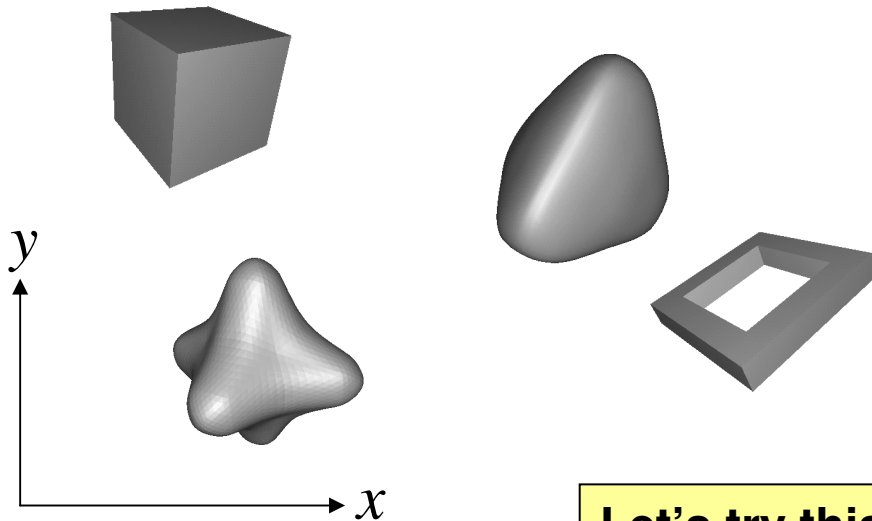
In 2D space



# 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:

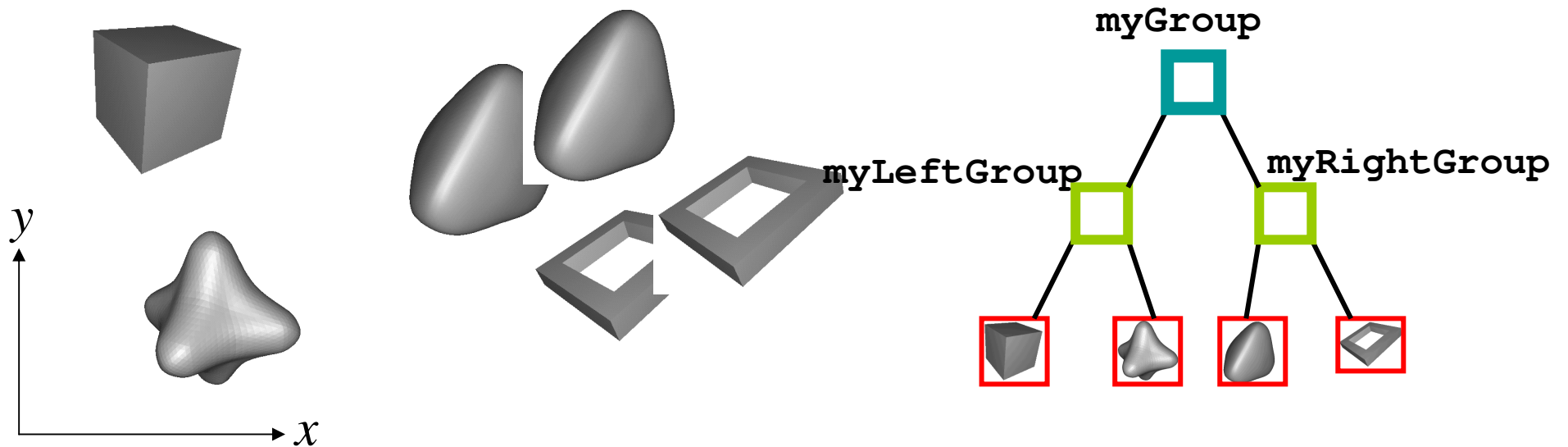


Let's try this:

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

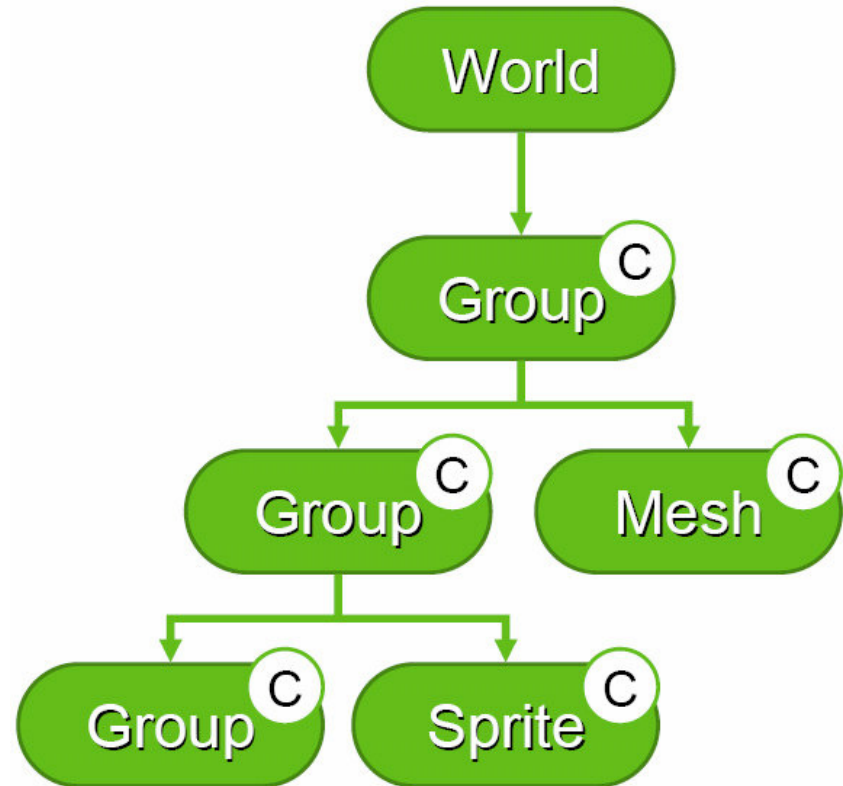
# Scene graph example (cont'd)

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

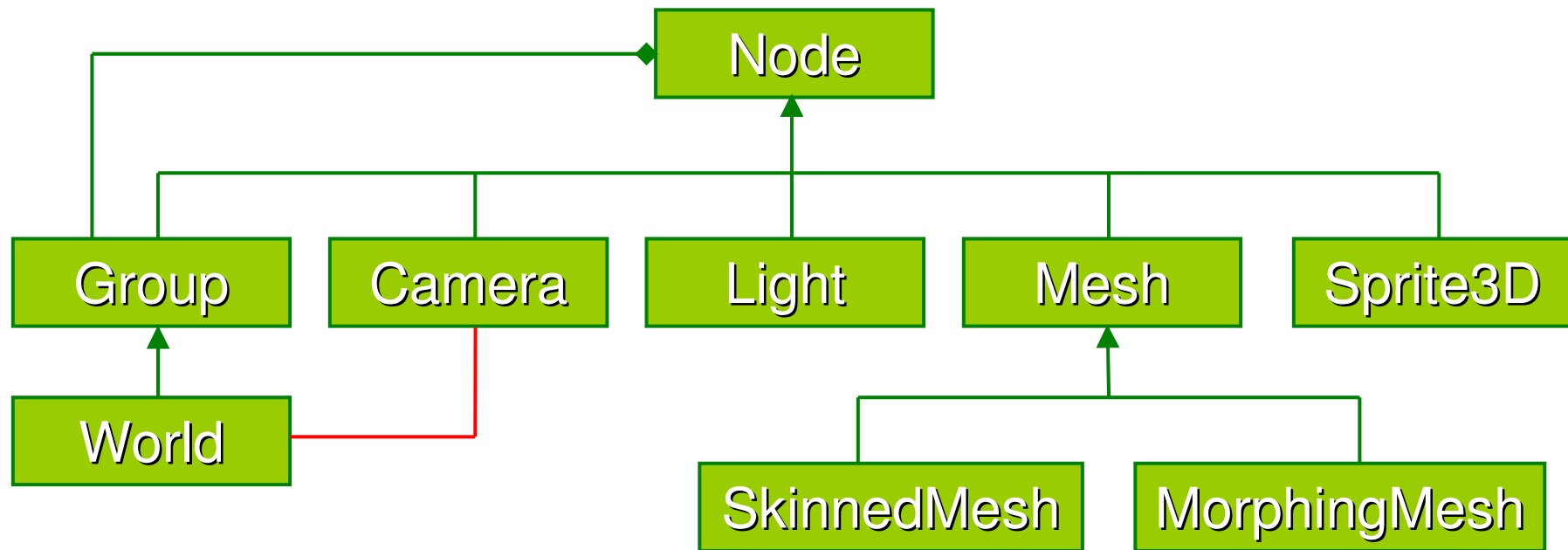


# Scene graph - transform

- Every node in the scene graph has a transform, composed of:
  - Translation, T
  - Rotation, R
  - Scale, S
  - Extra matrix, M (rarely needed)
- Final transform  
 $C = TRSM$
- Transforms are hierarchical



# Scene graph - classes

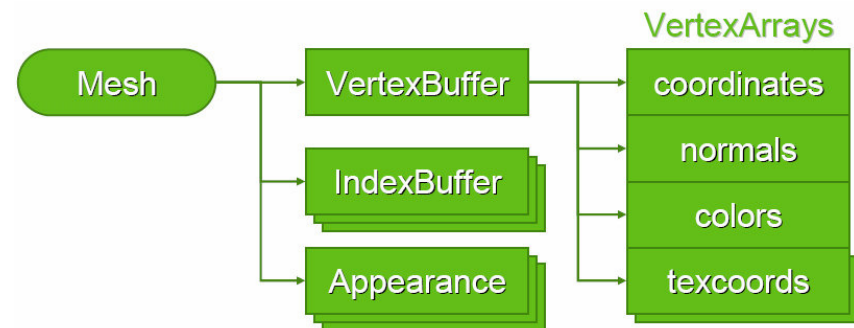
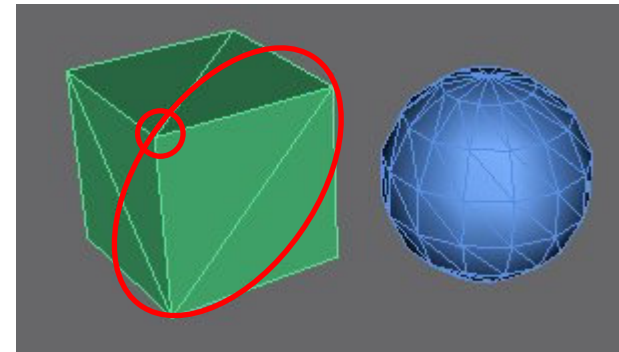


The World must have an "active" camera before it can be rendered

- See `World.setActiveCamera()`

# Meshes

- Consists of
  - Vertices
  - Faces
- In M3G
  - A single vertex buffer
    - Vertex positions
    - Attributes
  - Index buffers
    - Faces
  - Appearances
    - The appearance of a face



# Vertex Buffers

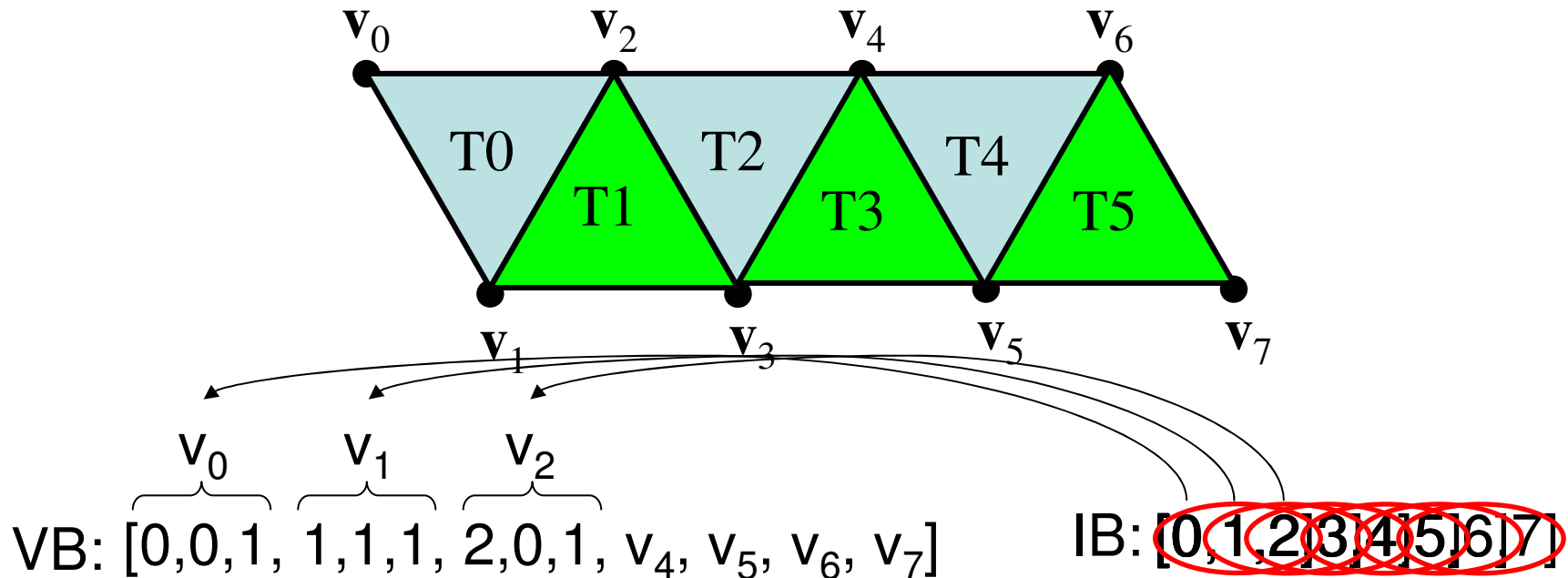
	8-bit	16-bit	32-bit	Float	2D	3D	4D
Vertices	✓	✓	✗	✗	✗	✓	✗
Texcoords	✓	✓	✗	✗	✓	✓	✗
Normals	✓	✓	✗	✗		✓	
Colors	✓		✗	✗		✓	✓

- Only integer values
- Floating point bias and scale values can be used for positions and texcoords.

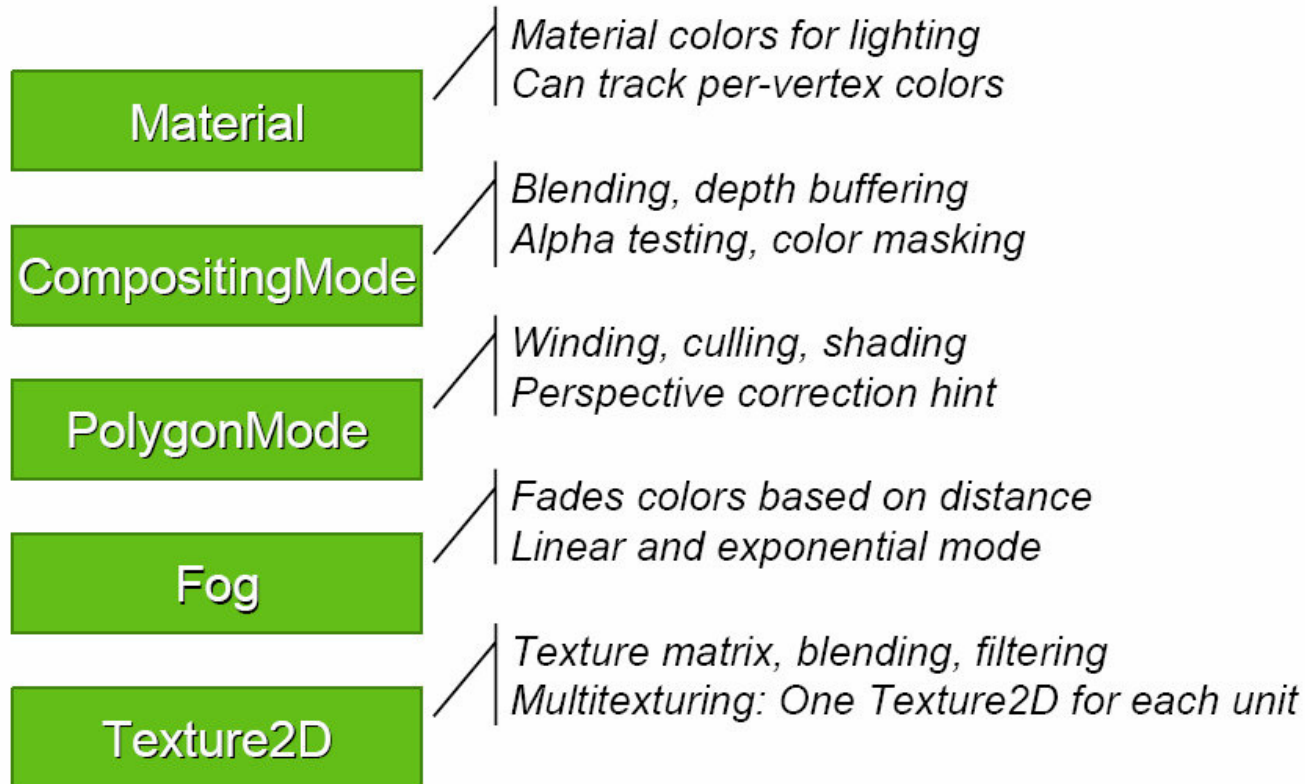
$$p' = p * \text{scale} + \text{bias}$$

# Index Buffers

- Start with a vertex buffer
- Create triangles connecting the vertices
  - Using a *triangle strip*

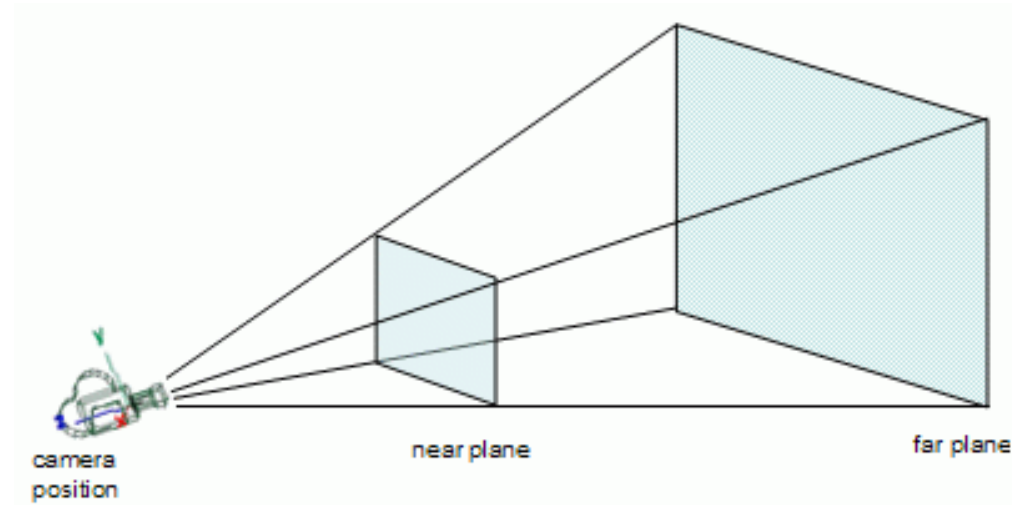


# Appearance



- You can use the default appearance in the assignment
  - Renders the object with specified vertex colors

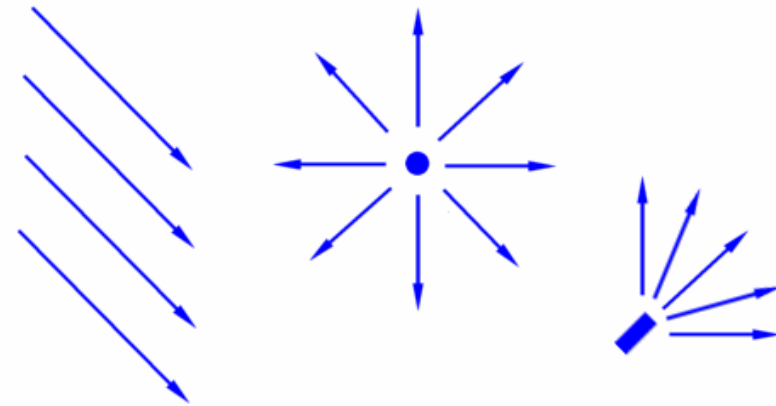
# Camera



- Virtual camera observing the scene
  - Looks down negative z
  - Projection is set as fov, aspect ratio, near and far clip planes.

# Lights

- Same type of lights as in OpenGL
  - Ambient, Directional, Omni (Point light), Spot
- Position / direction is given by the light's transform
- Functions to set intensity, color, attenuation.



# Assignment 1

1. Load and render an existing scene
2. Create your own mesh (in code), and build a simple scene graph.
3. Make an asteroids game using your meshes
4. Write a mesh loader, and use the given meshes. Add some lighting

# Assignment 1 – The Code

- `public void startApp()`
  - Called when the application is started.
  - Put your loading / scene graph setup code here
- `public void paint(Graphics g)`
  - Called periodically to repaint the canvas
  - Put your render and animation code here