CS193P - Lecture 19

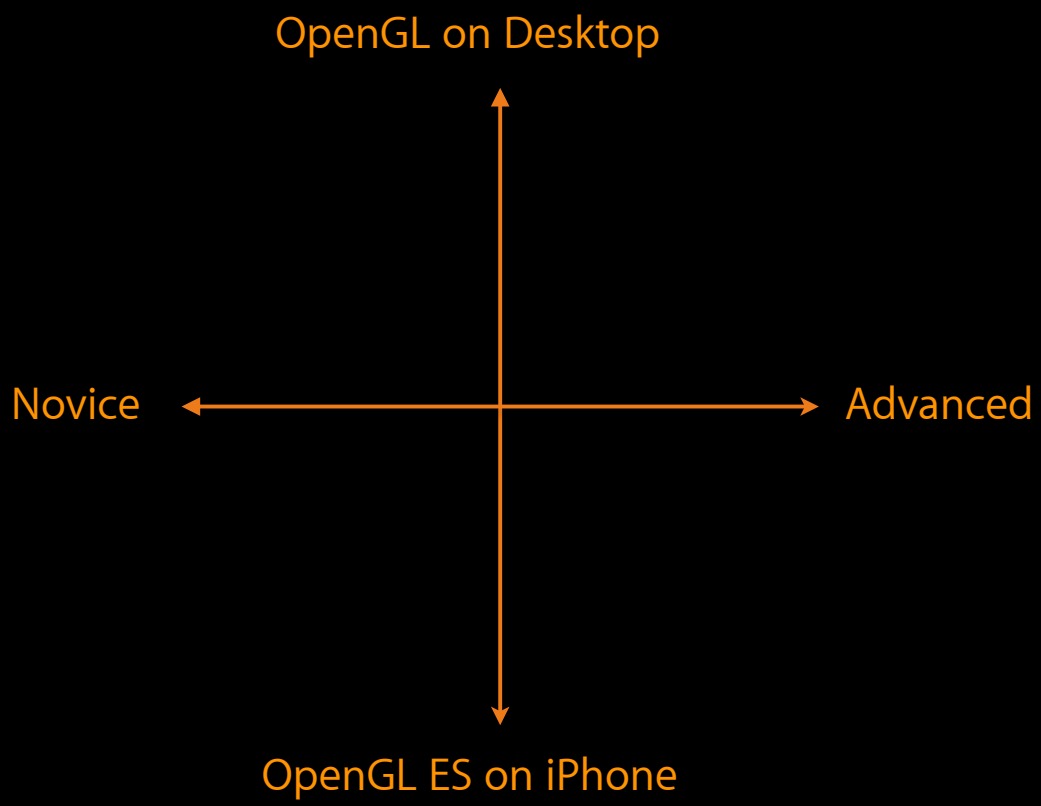
iPhone Application Development

OpenGL ES

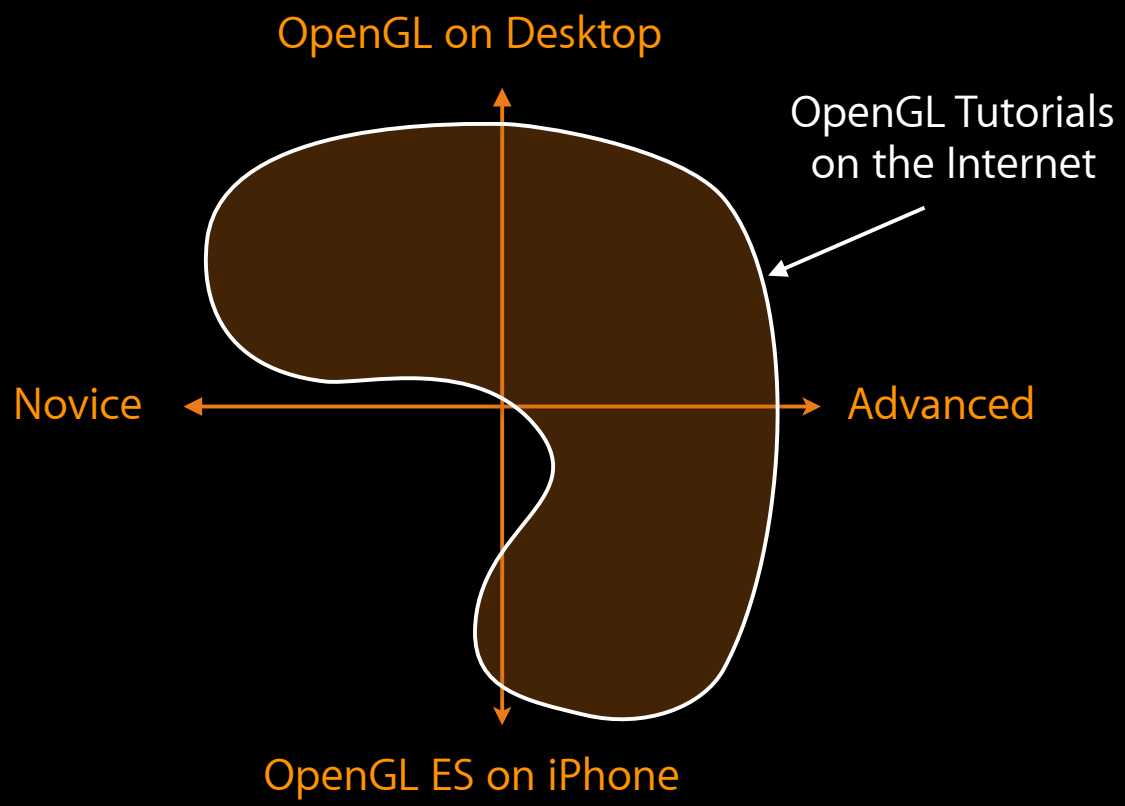
Announcements

- **Final projects** due in 7 days
  - Tuesday, March 16th 11:59 pm
  - Submit:
    - Code
    - Power-point/Keynote slides
    - ReadMe file
- Final project **demos**
  - March 18, from 12:15 - 3:15 pm in Hewlett 201
  - 2 minute presentation, followed by demo-fair
    - Rapid-fire!!
    - Time limit strictly enforced
  - Let us know if you do not want to be recorded

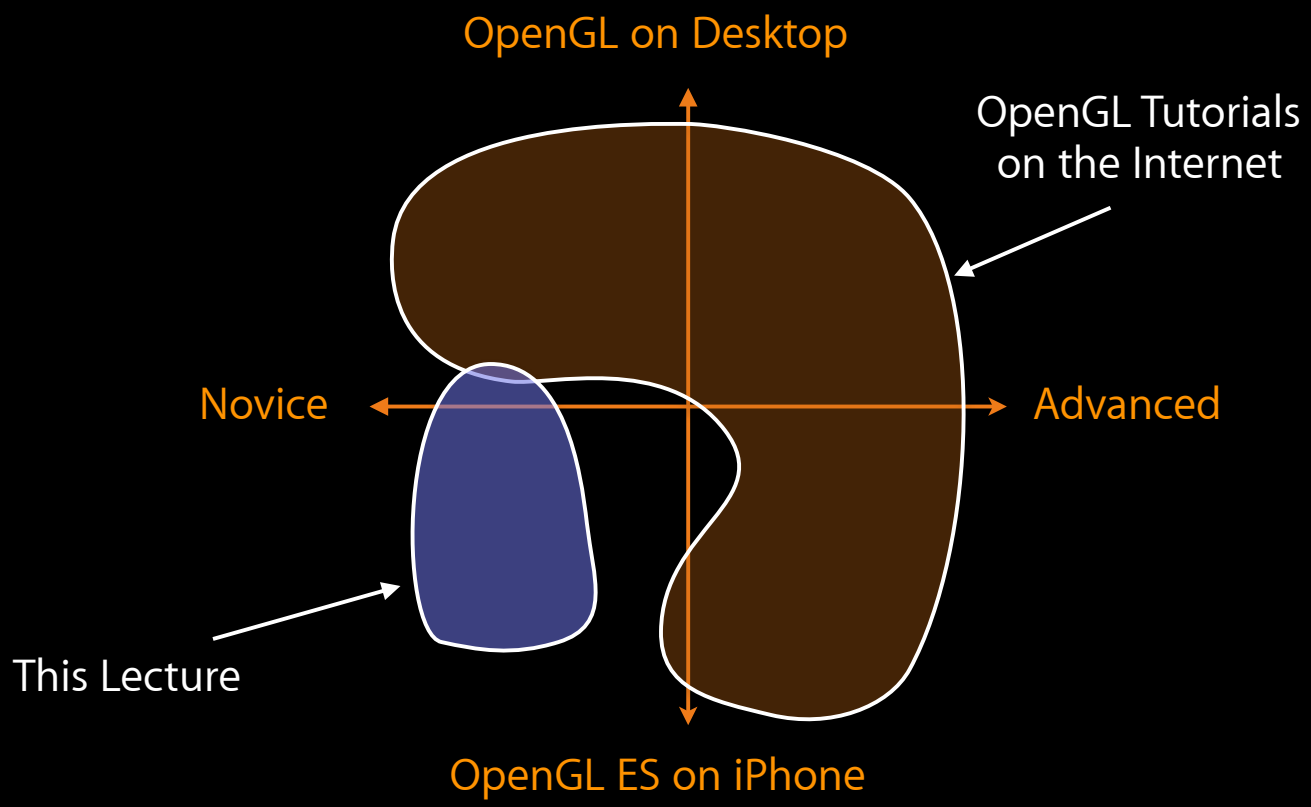
Today's Topics



Today's Topics



Today's Topics



Today's Topics

- OpenGL Overview
- Coordinate Systems and Transformations
- Drawing Geometry
- Using Textures
- Other Details

OpenGL Overview

OpenGL Overview

OpenGL Overview

- Software interface for graphics hardware

OpenGL Overview

- Software interface for graphics hardware
- Quickly render 2D or 3D graphics

OpenGL Overview

- Software interface for graphics hardware
- Quickly render 2D or 3D graphics
- Hardware implementation agnostic

OpenGL Overview

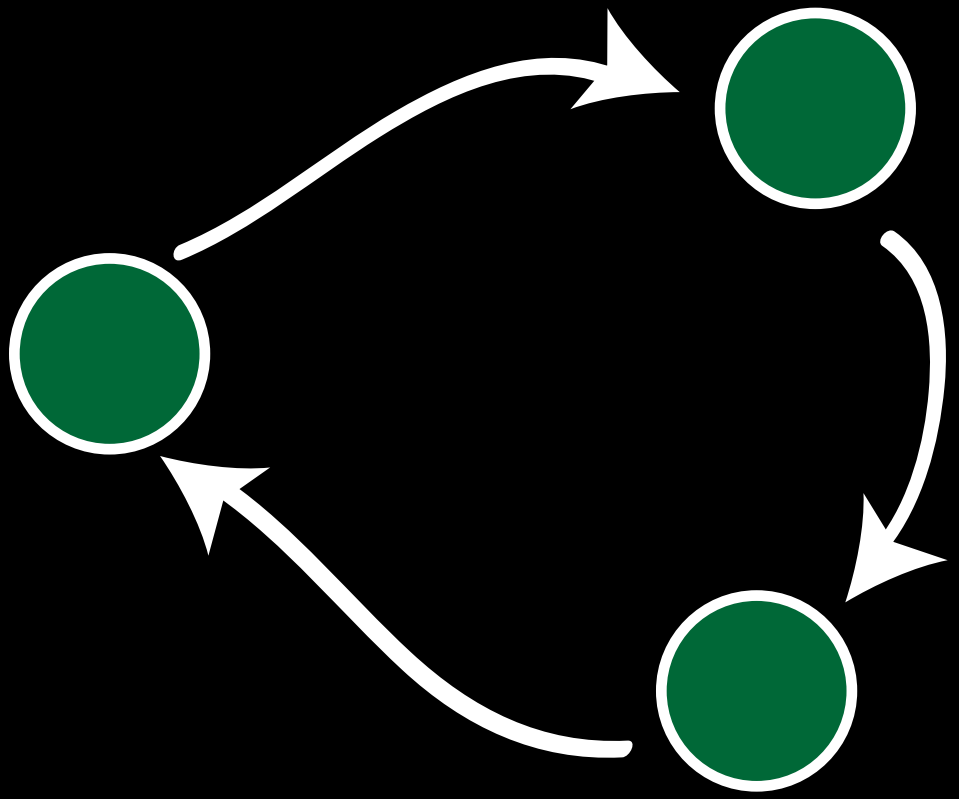
- Software interface for graphics hardware
- Quickly render 2D or 3D graphics
- Hardware implementation agnostic

OpenGL Overview

- Software interface for graphics hardware
- Quickly render 2D or 3D graphics
- Hardware implementation agnostic

- OpenGL ES is a subset of OpenGL
  - GLUT and GLU are not available on the iPhone

OpenGL is a state machine



OpenGL is a state machine

- Change Machine State

```
glEnable(); glDisable(); glMatrixMode(); glBindFramebufferOES();
glViewport(); glVertexPointer(); glColorPointer(); glTranslatef() ...
```

- Issue Drawing Commands

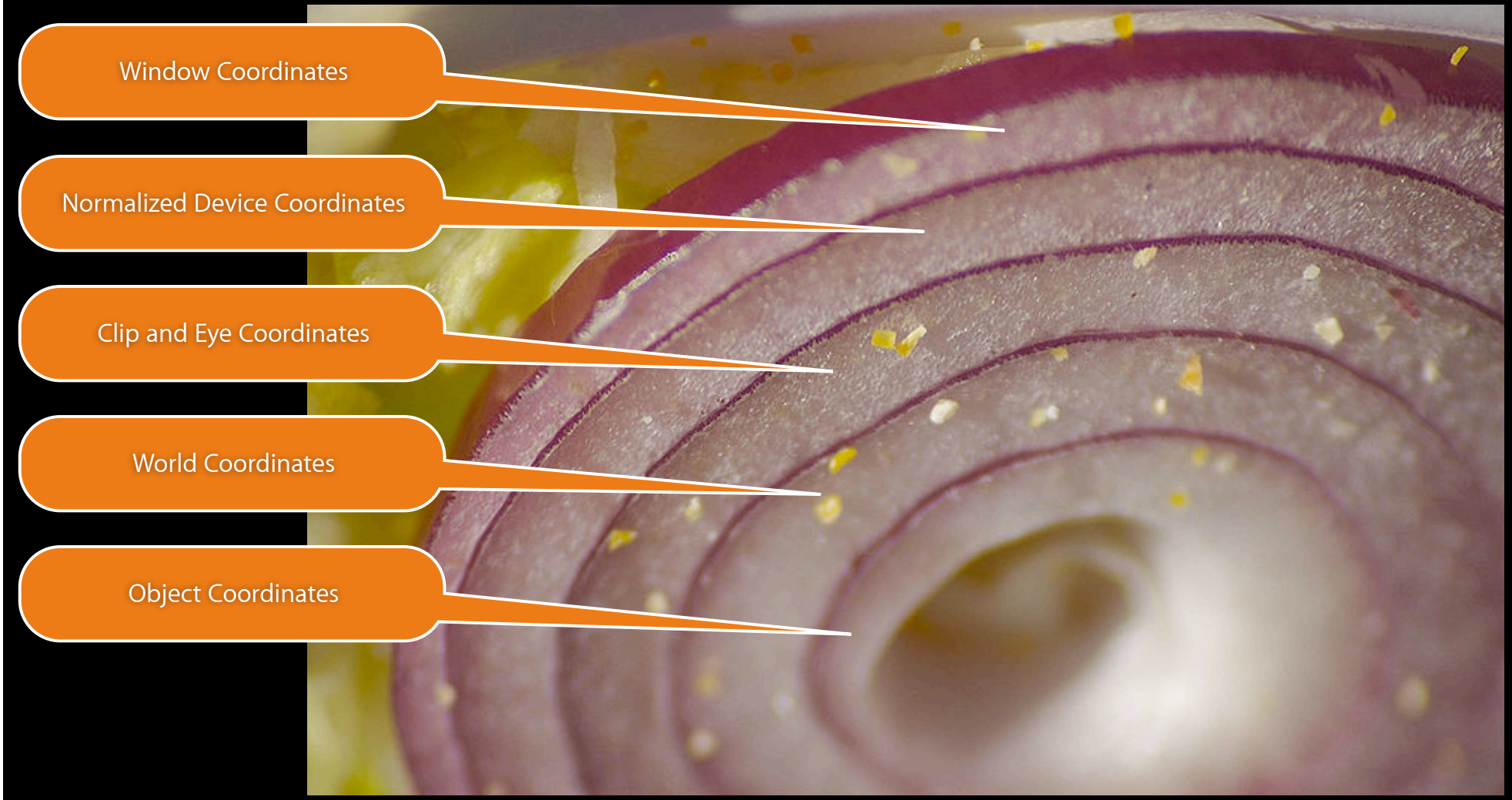
```
glDrawArrays(); glDrawElements(); ...
```

- Read Back State, Drawing Results

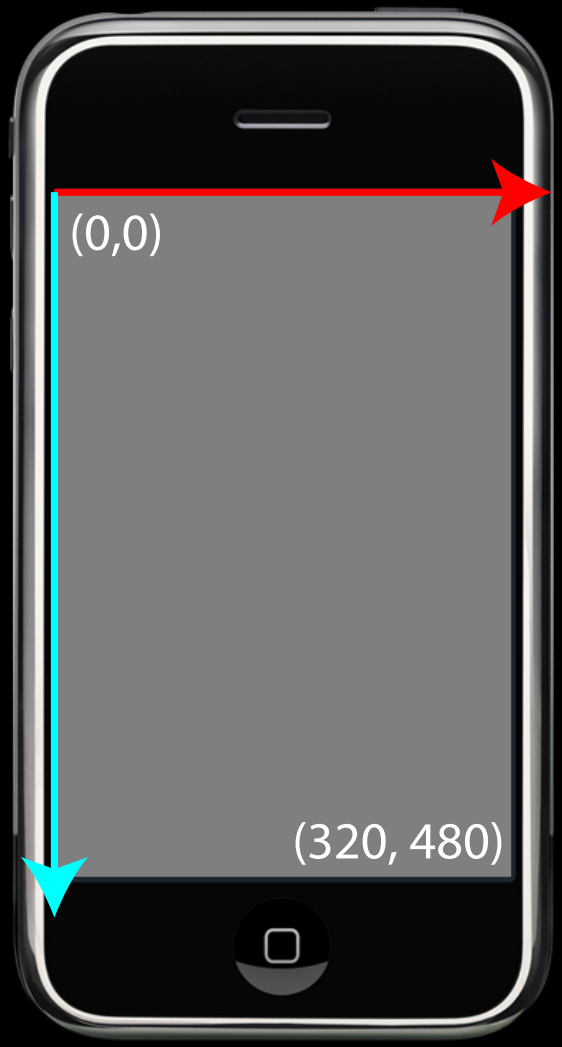
```
glGetBooleanv(); glGetFloatv(); glReadPixels(); ...
```

Coordinate Systems

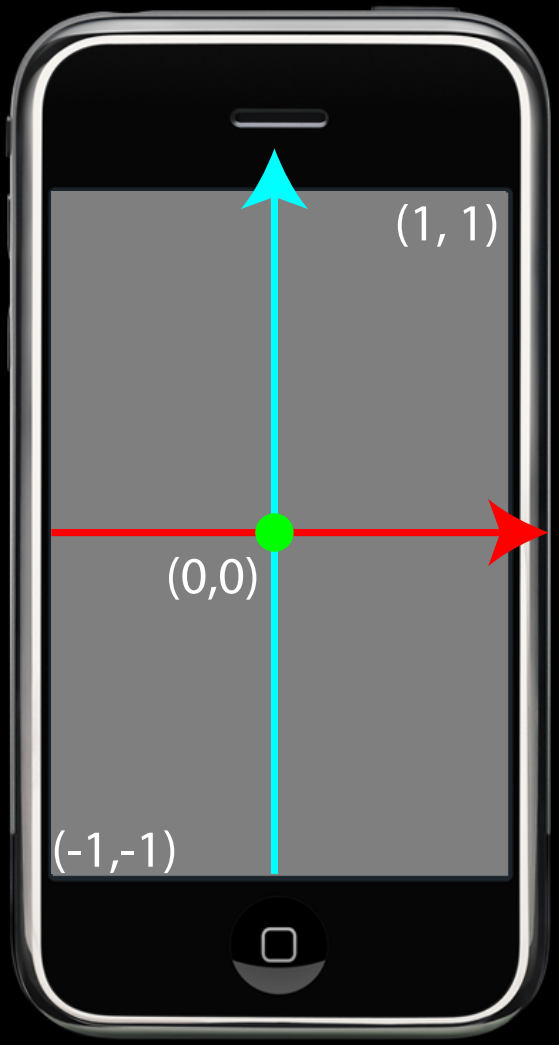
The Coordinate System Onion



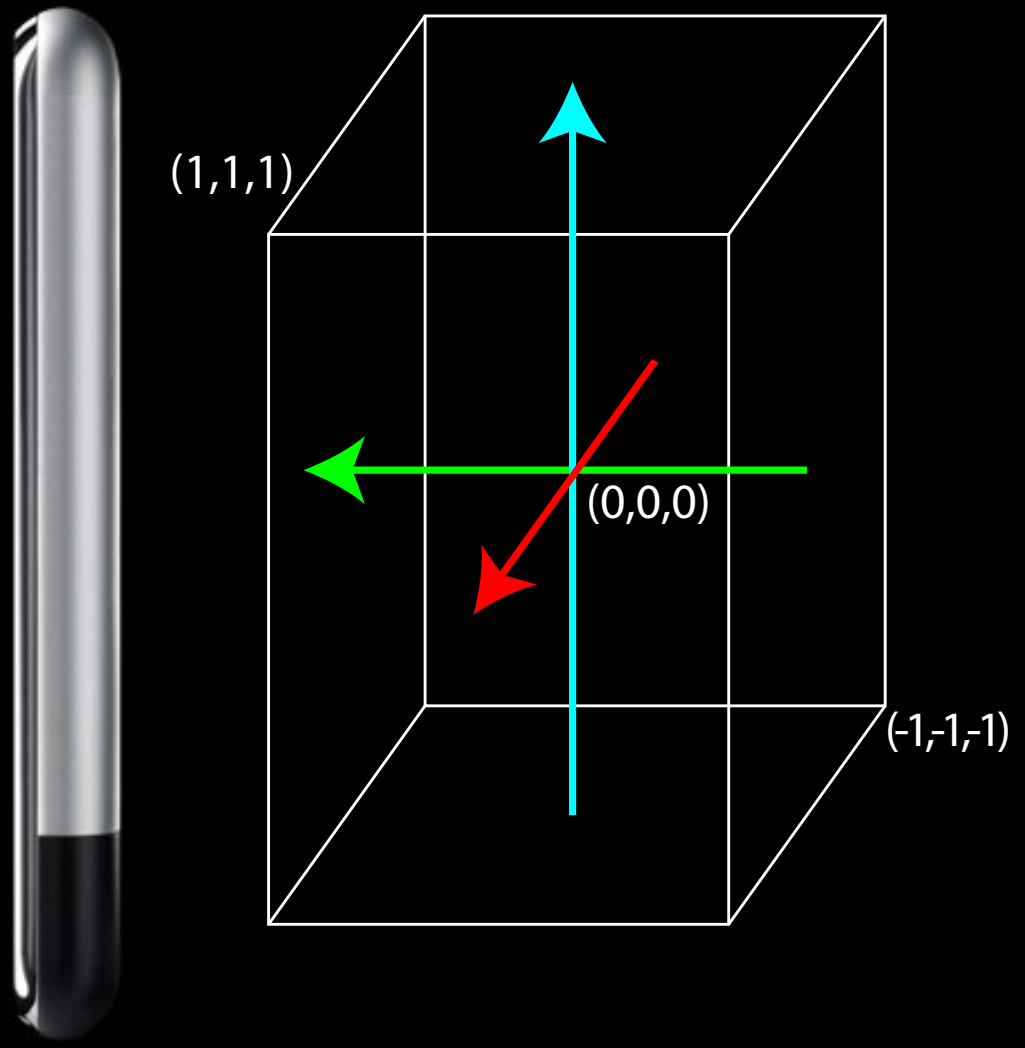
Window Coordinates



Normalized Device Coordinates



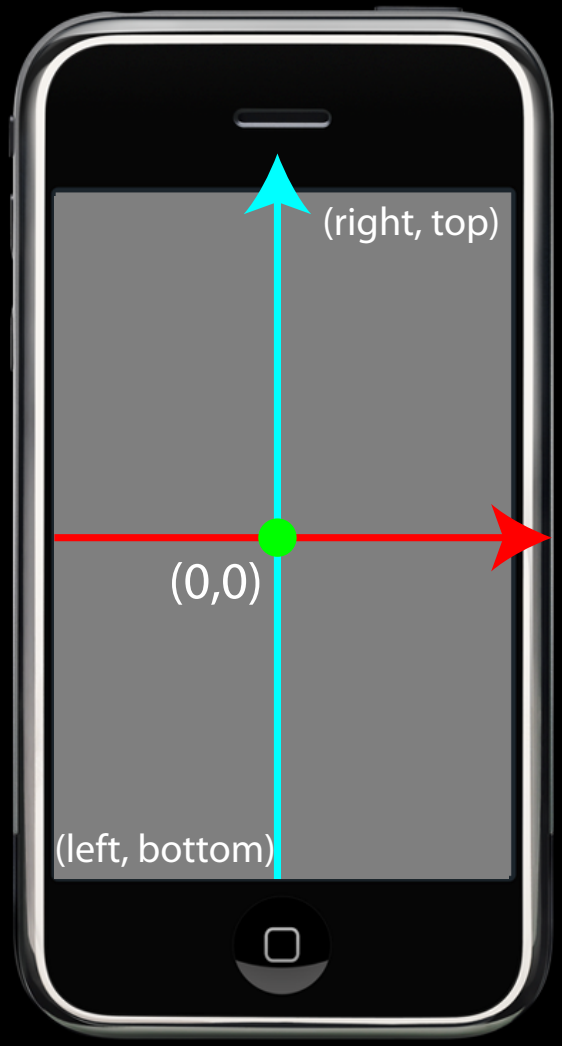
Normalized Device Coordinates



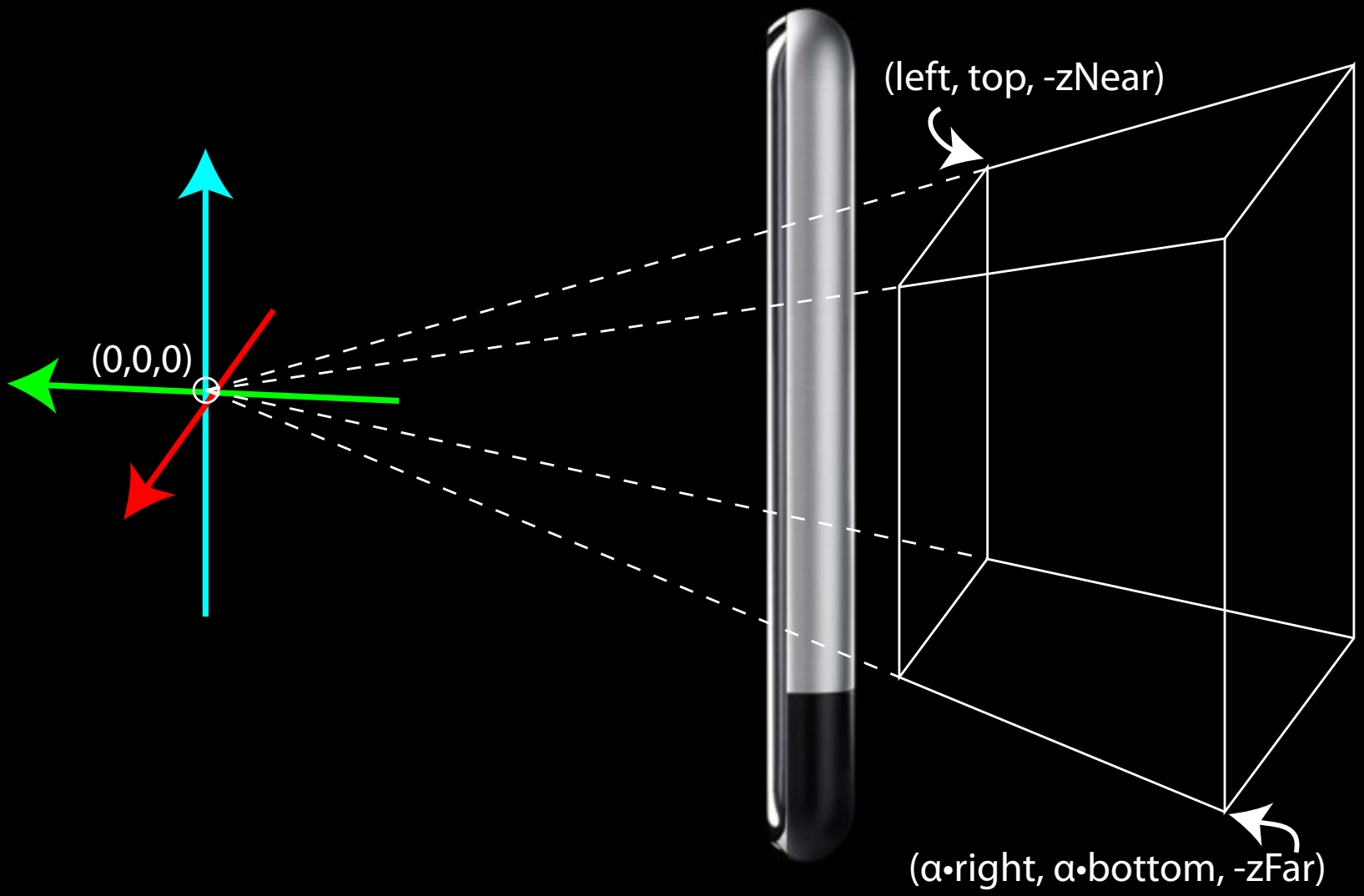
Clip Coordinates

ILLUSTRATION
NOT FOUND

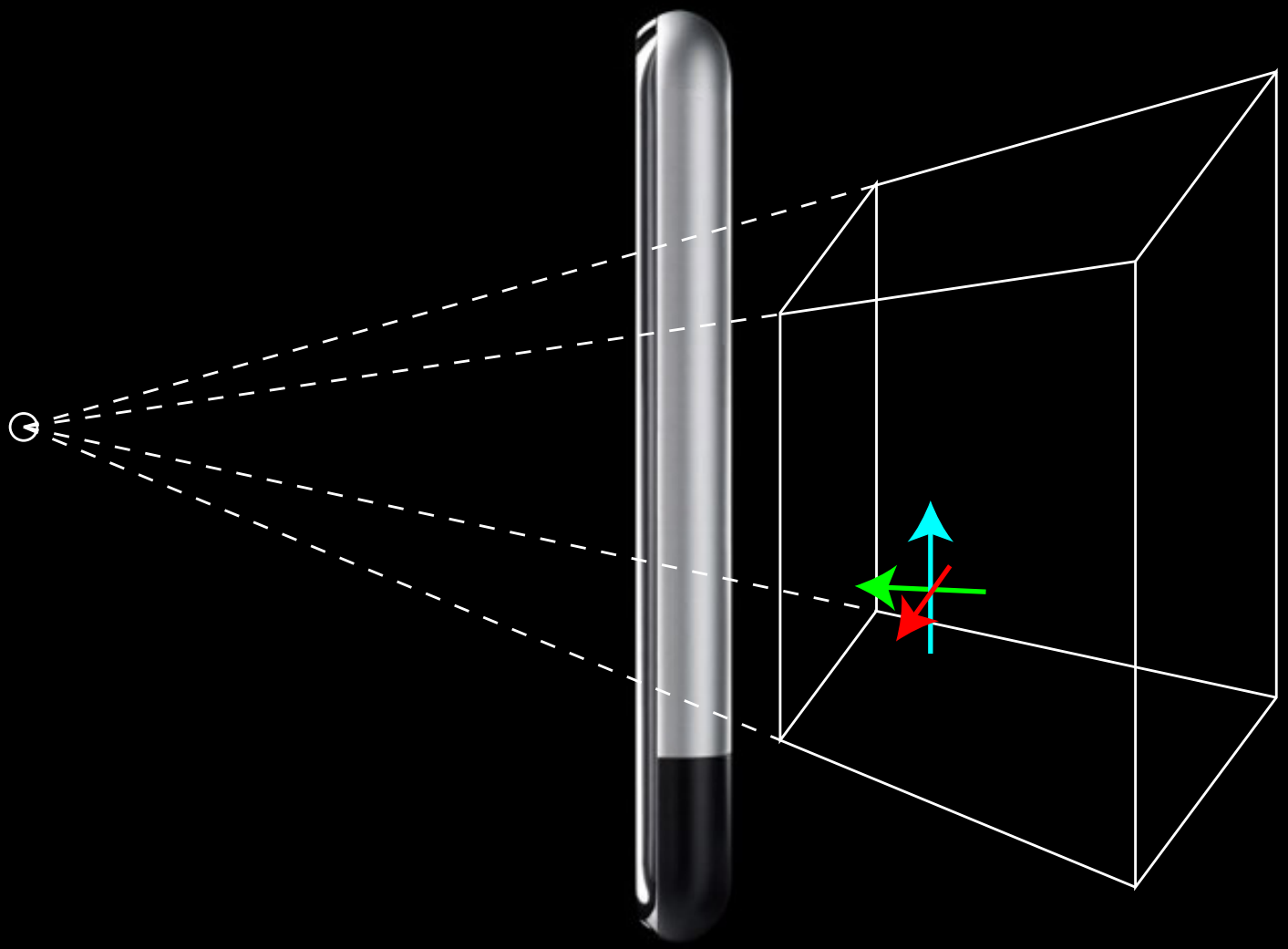
Eye Coordinates



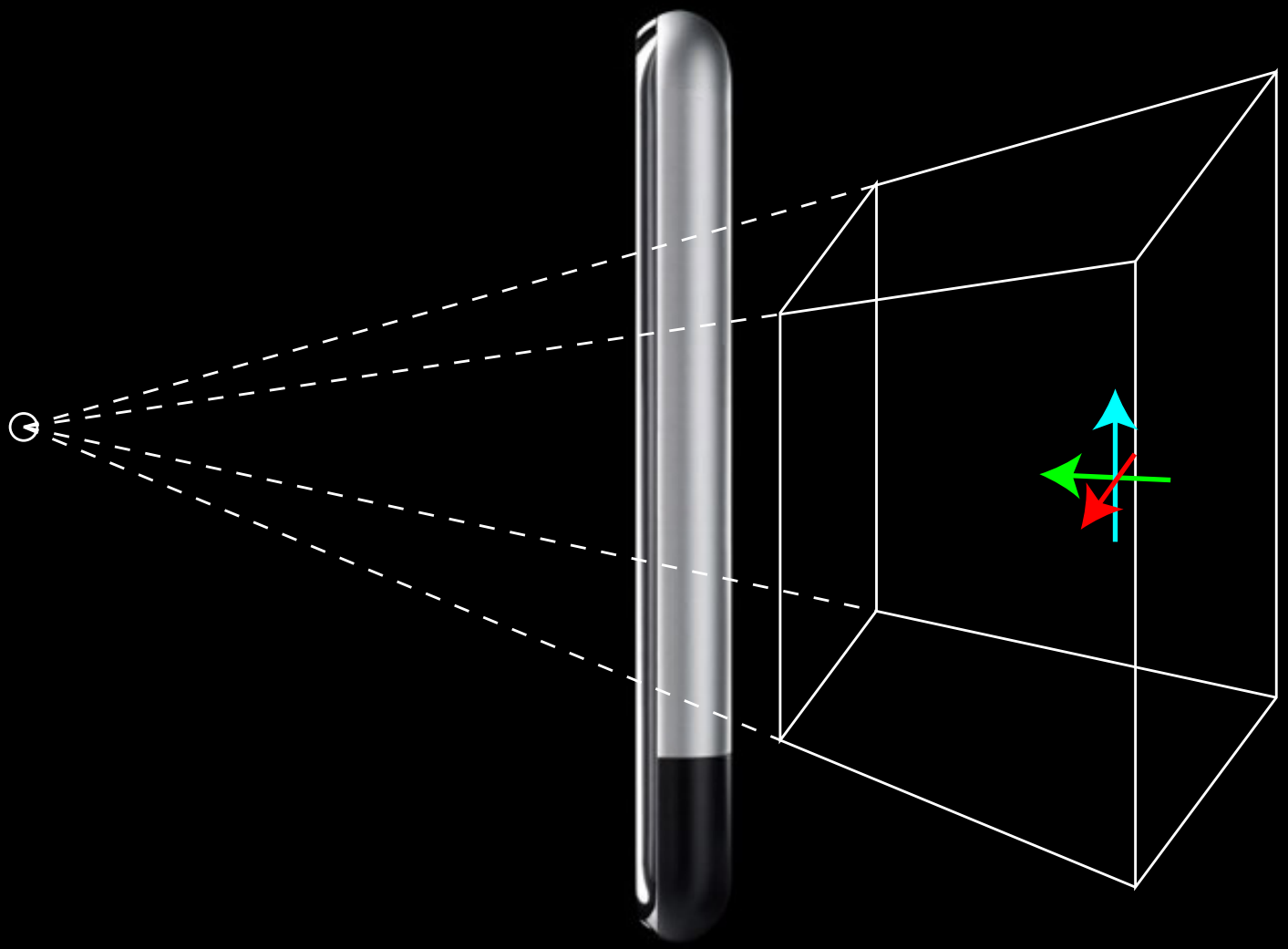
Eye Coordinates



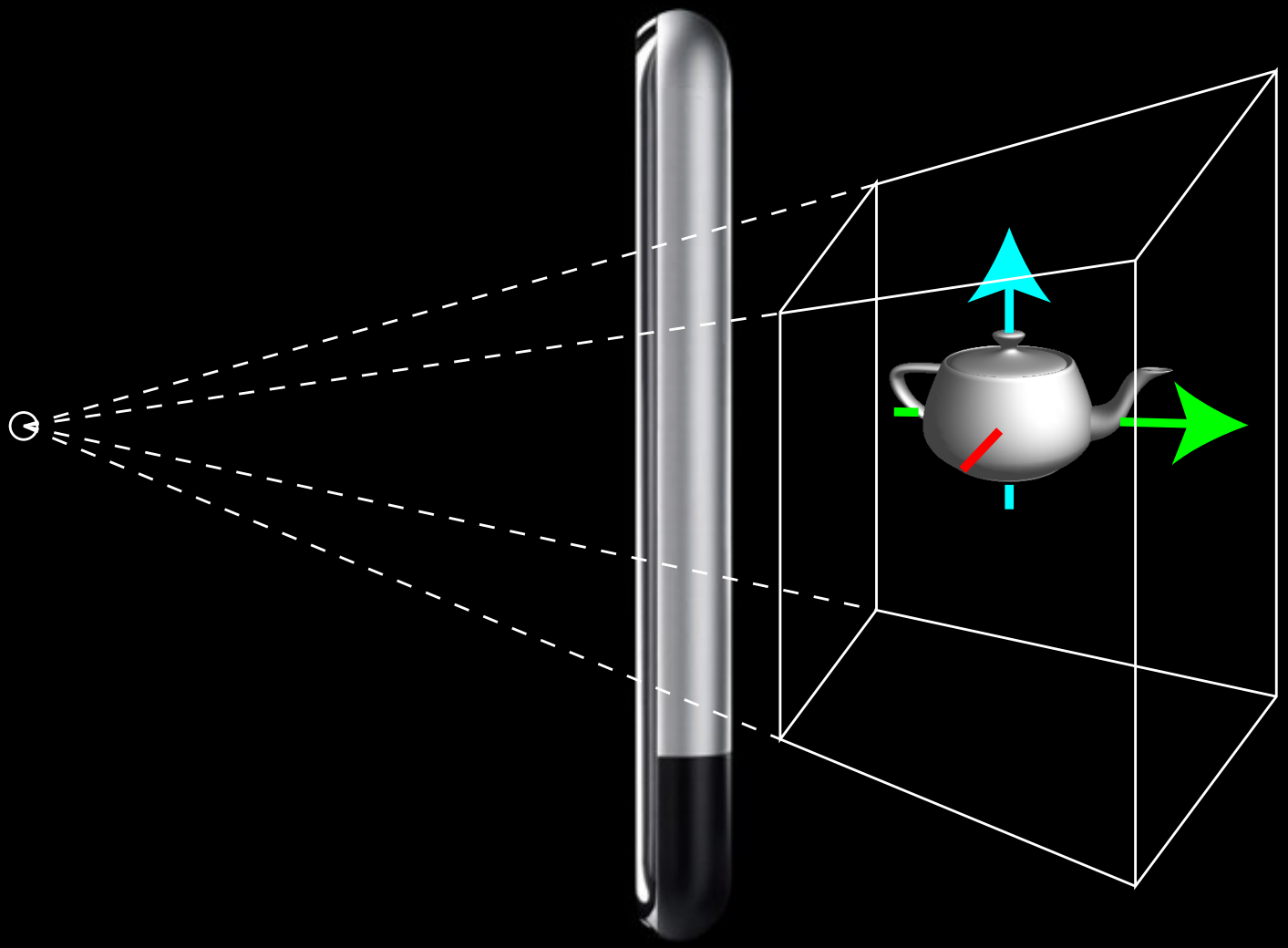
World Coordinates



World Coordinates



Object Coordinates



Demo

Transformations

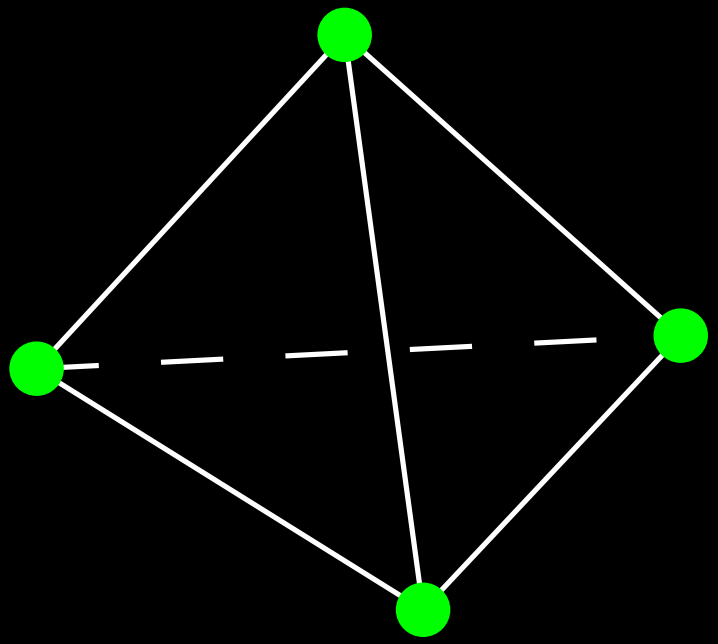
Transformations Cheat Sheet

```
glMatrixMode(GL_PROJECTION);
glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glOrthof(left, right, bottom, top, zNear, zFar);
glFrustumf(left, right, bottom, top, zNear, zFar);

glRotatef(degrees, x, y, z);
glTranslatef(x, y, z);
glScalef(x, y, z);
glMultMatrixf(matrix);

glPushMatrix();
glPopMatrix();
```

Drawing Geometry

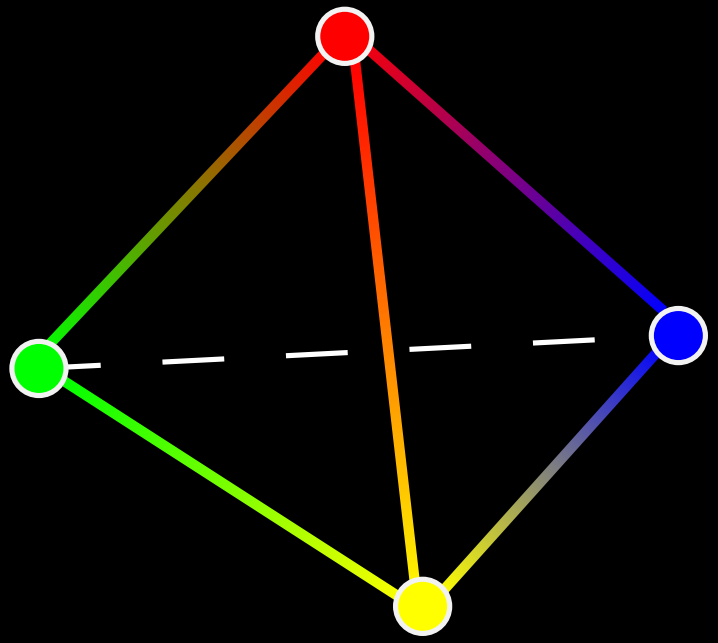
Vertices

- OpenGL ES draws triangles, lines, and points
- Object space vertices mapped into window space
- Rasterize the shapes to get pixels



Colors

- Each vertex can have a color associated
- RGBA
  - Alpha usually means opacity
- Lines and triangles interpolate colors



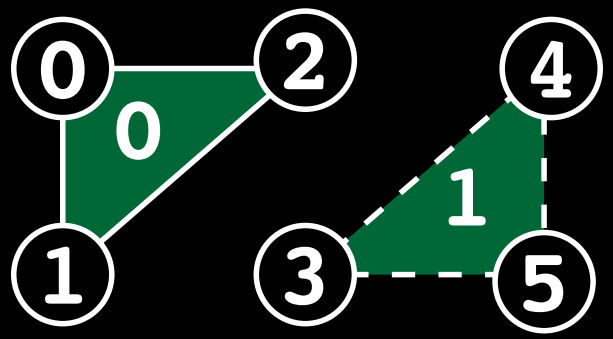
Drawing Modes

- Vertices are passed to OpenGL ES in arrays
- Drawing modes determine how vertices are interpreted to produce shapes
- Vertex order matters

Drawing Modes

- Vertices are passed to OpenGL ES in arrays
- Drawing modes determine how vertices are interpreted to produce shapes
- Vertex order matters

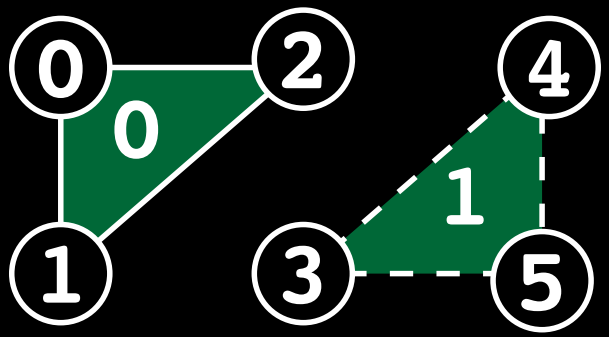
`GL_TRIANGLES`



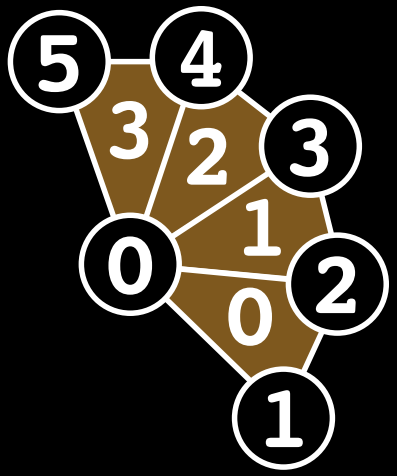
Drawing Modes

- Vertices are passed to OpenGL ES in arrays
- Drawing modes determine how vertices are interpreted to produce shapes
- Vertex order matters

GL_TRIANGLES



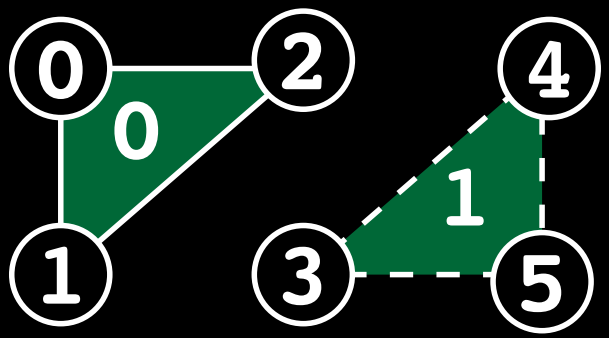
GL_TRIANGLE_FAN



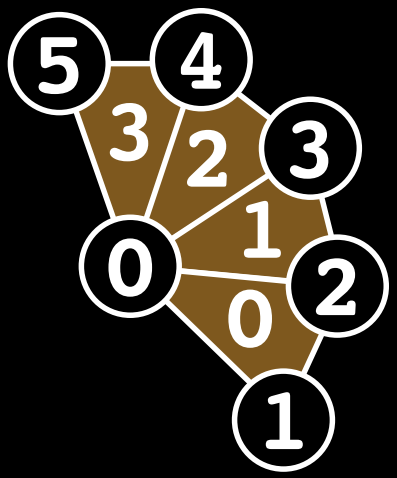
Drawing Modes

- Vertices are passed to OpenGL ES in arrays
- Drawing modes determine how vertices are interpreted to produce shapes
- Vertex order matters

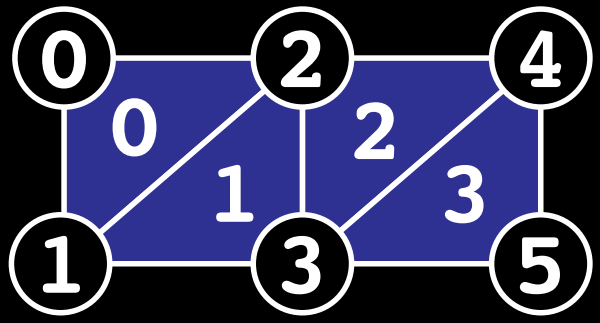
GL_TRIANGLES



GL_TRIANGLE_FAN

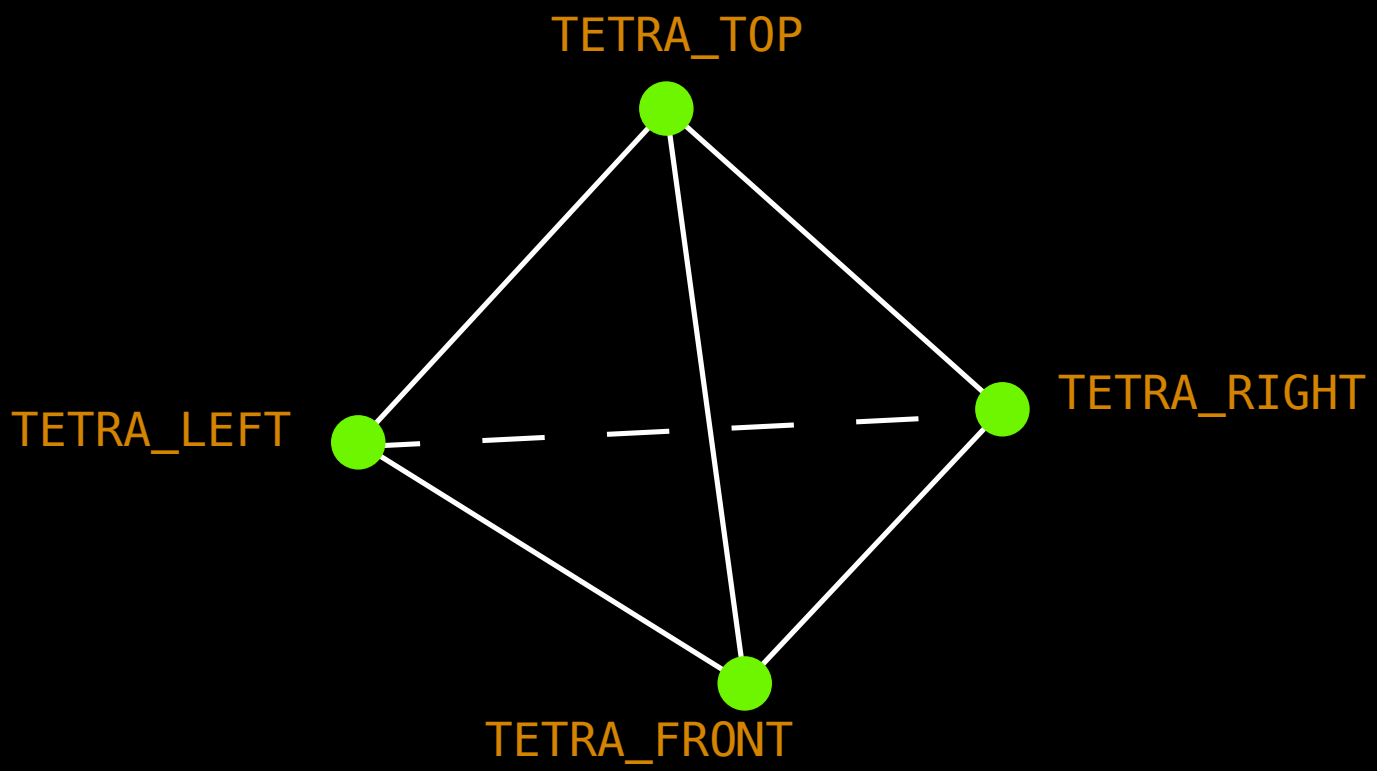


GL_TRIANGLE_STRIP



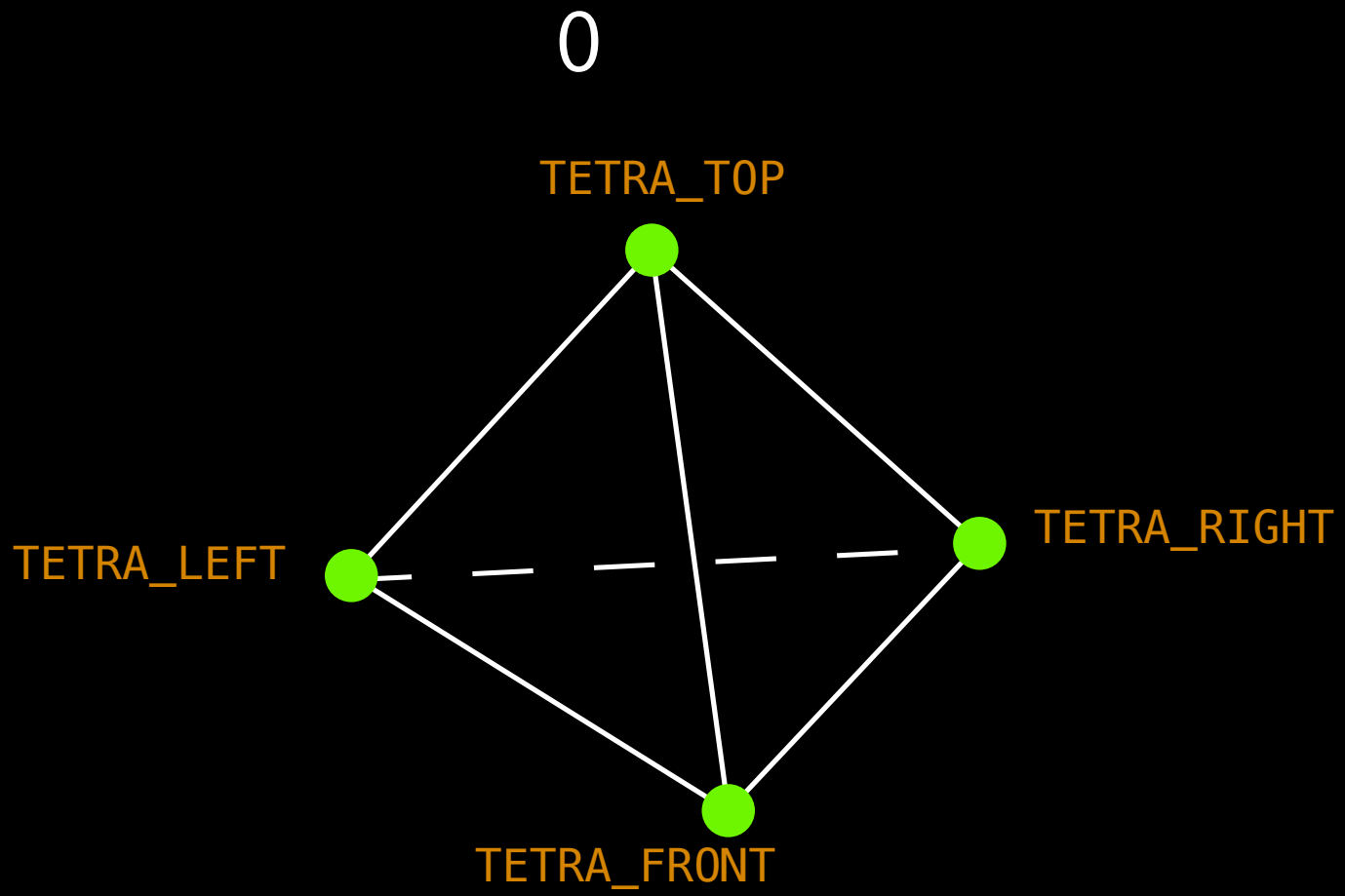
Drawing a Tetrahedron

- Use a triangle strip



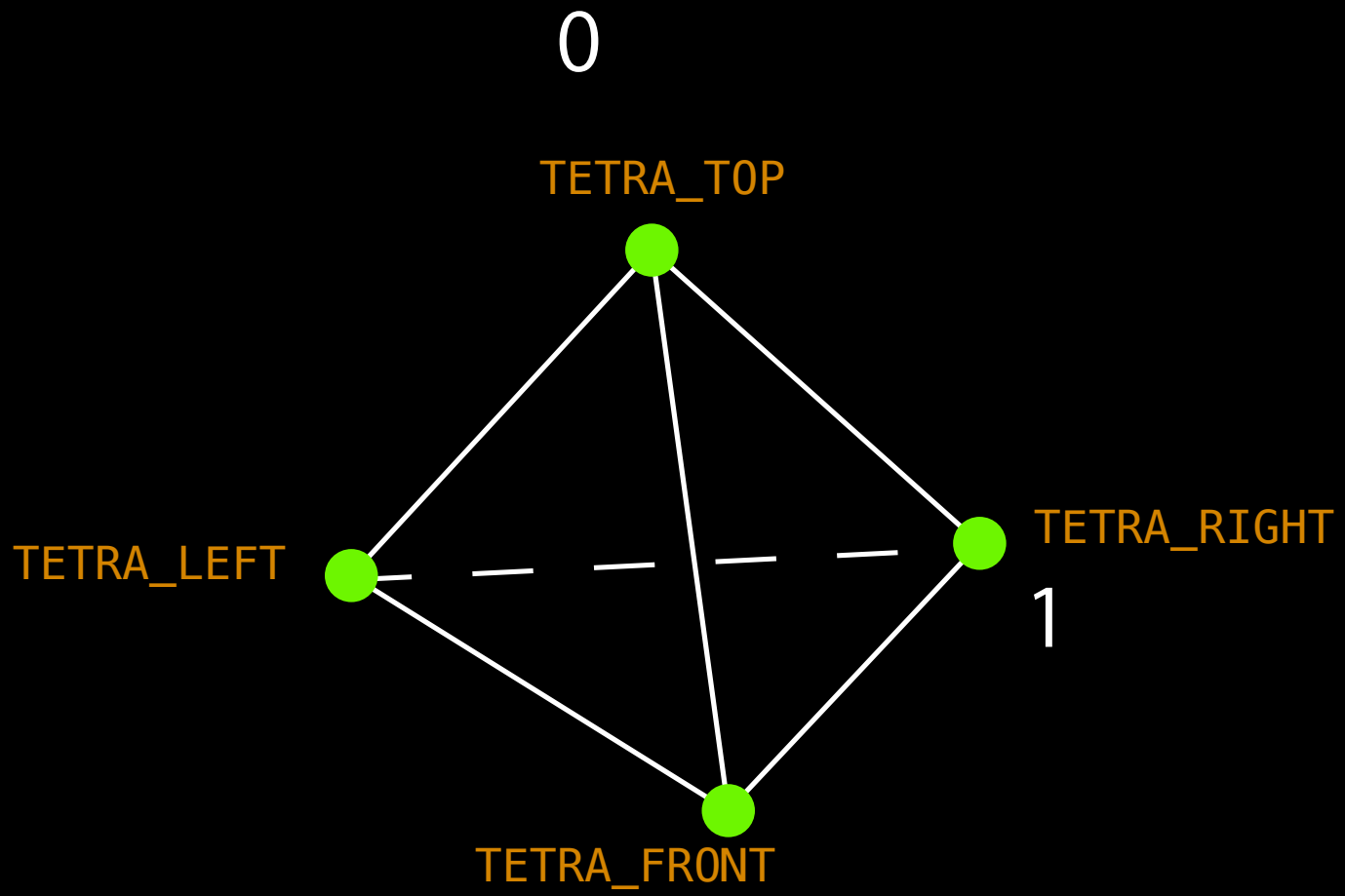
Drawing a Tetrahedron

- Use a triangle strip



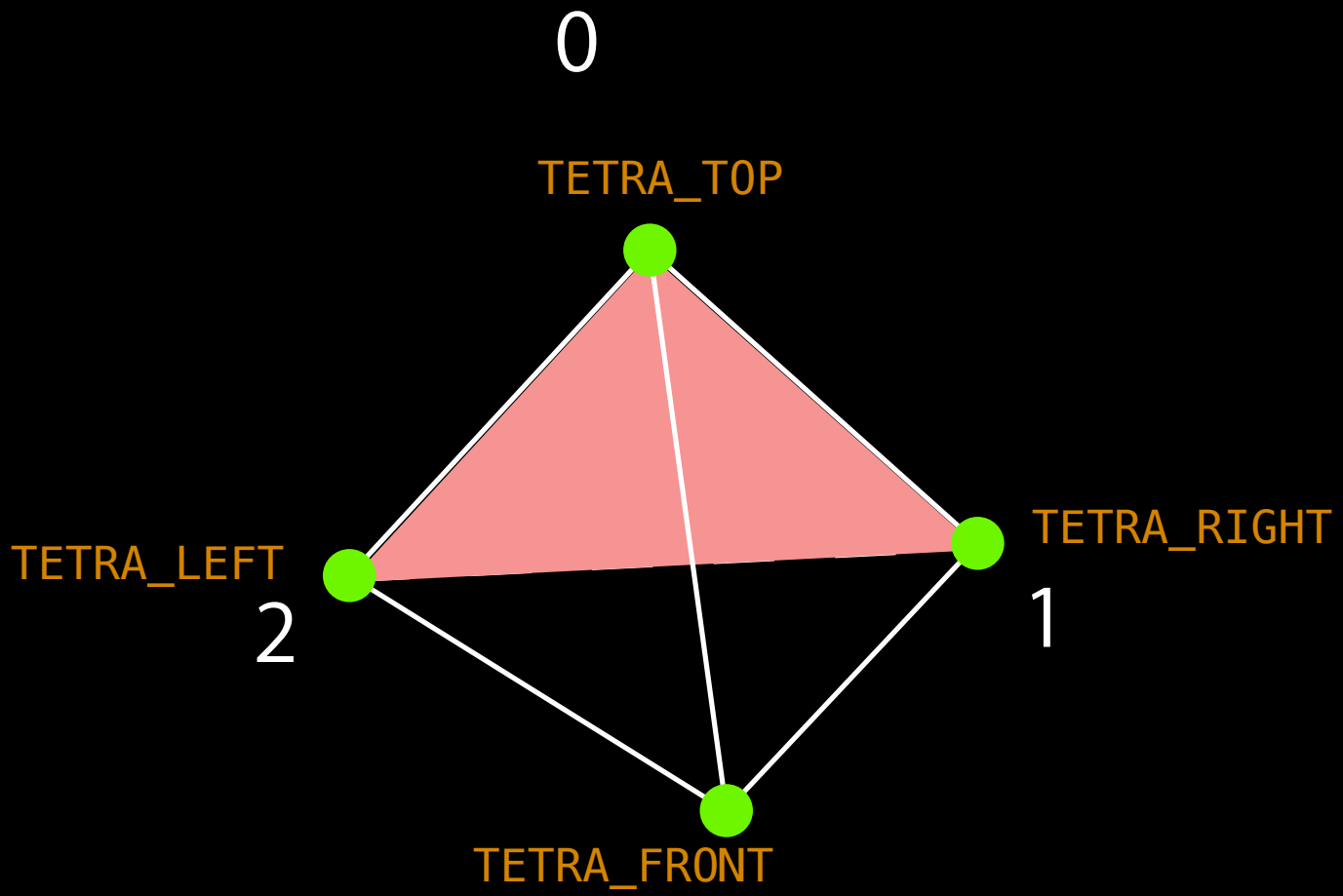
Drawing a Tetrahedron

- Use a triangle strip



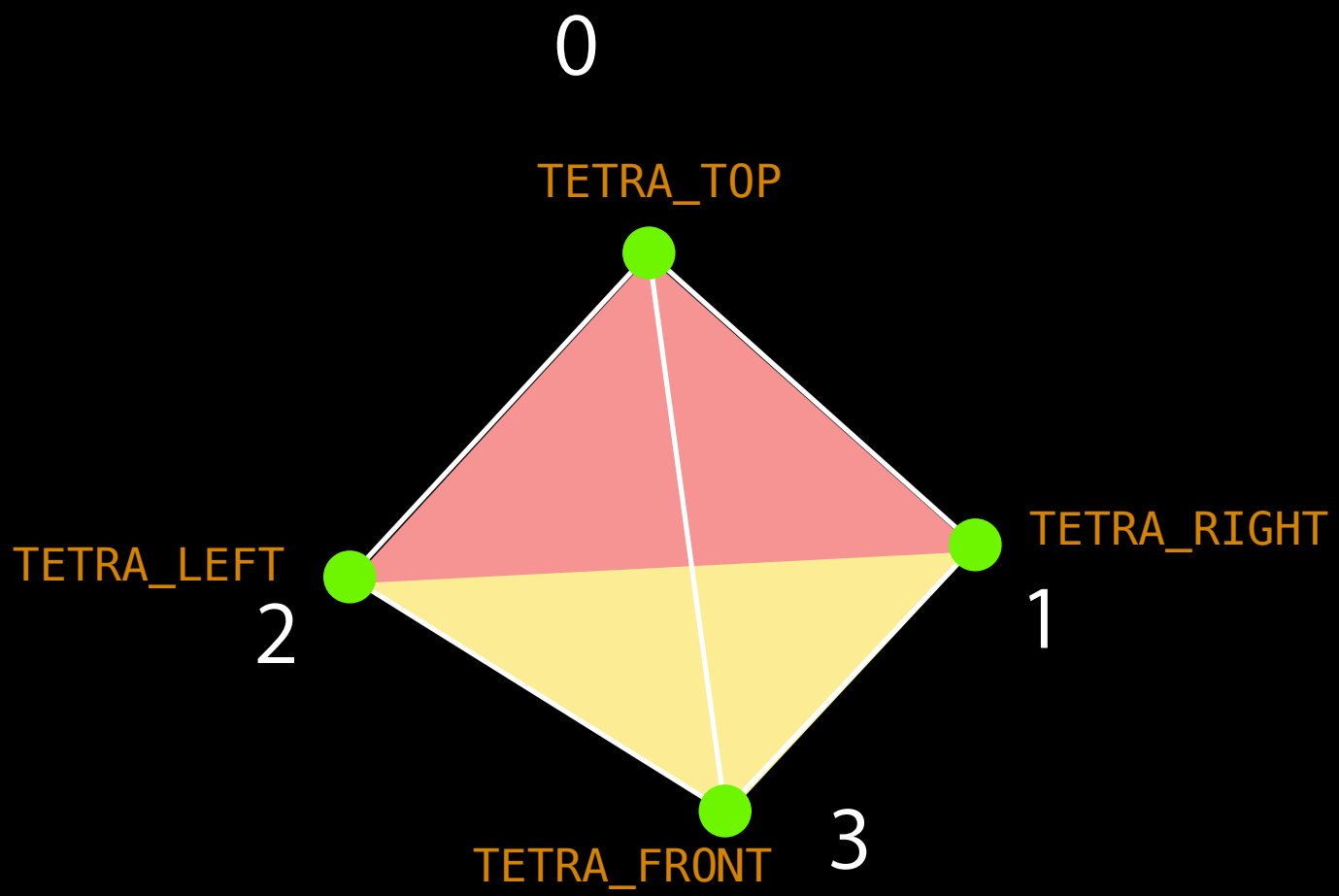
Drawing a Tetrahedron

- Use a triangle strip



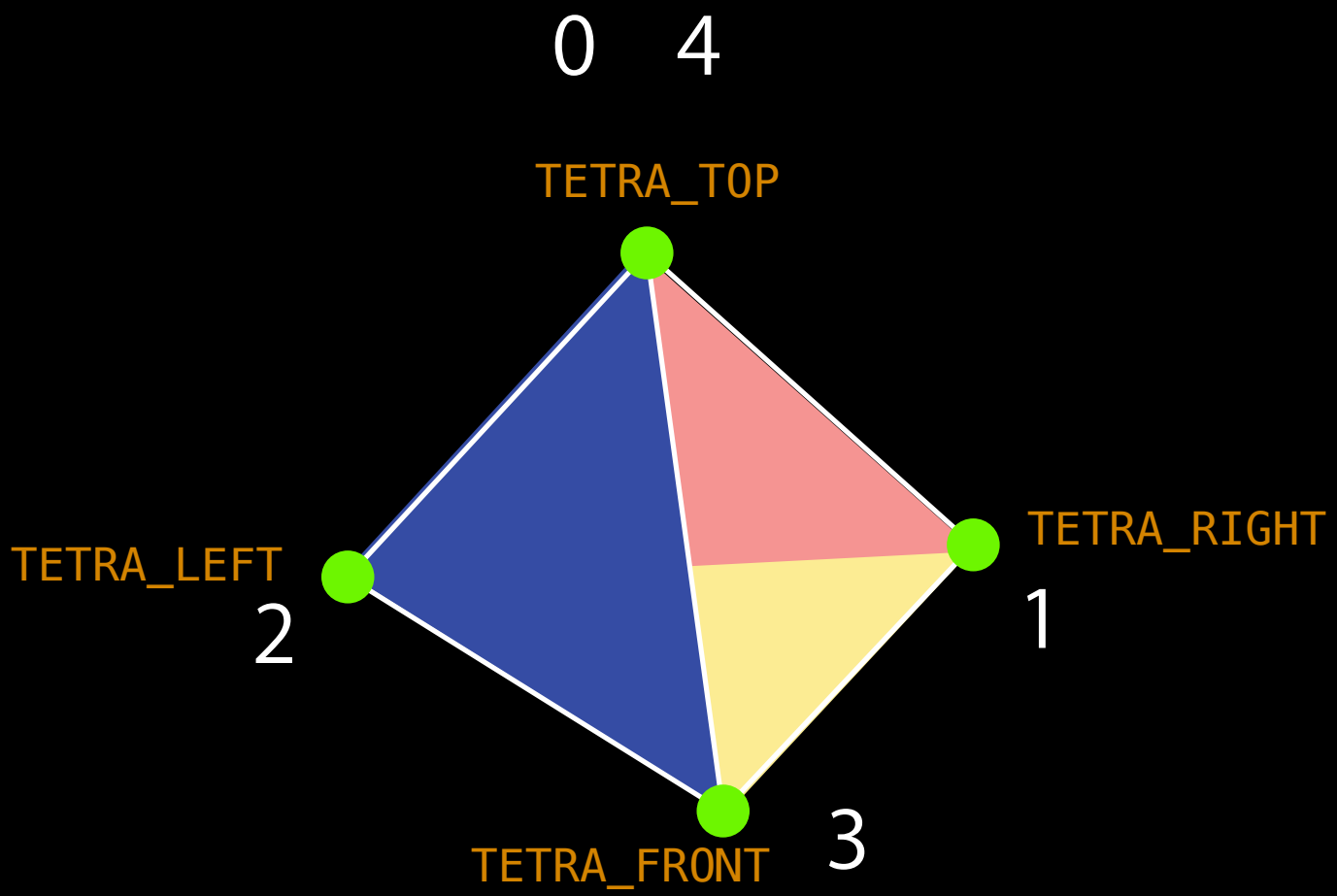
Drawing a Tetrahedron

- Use a triangle strip



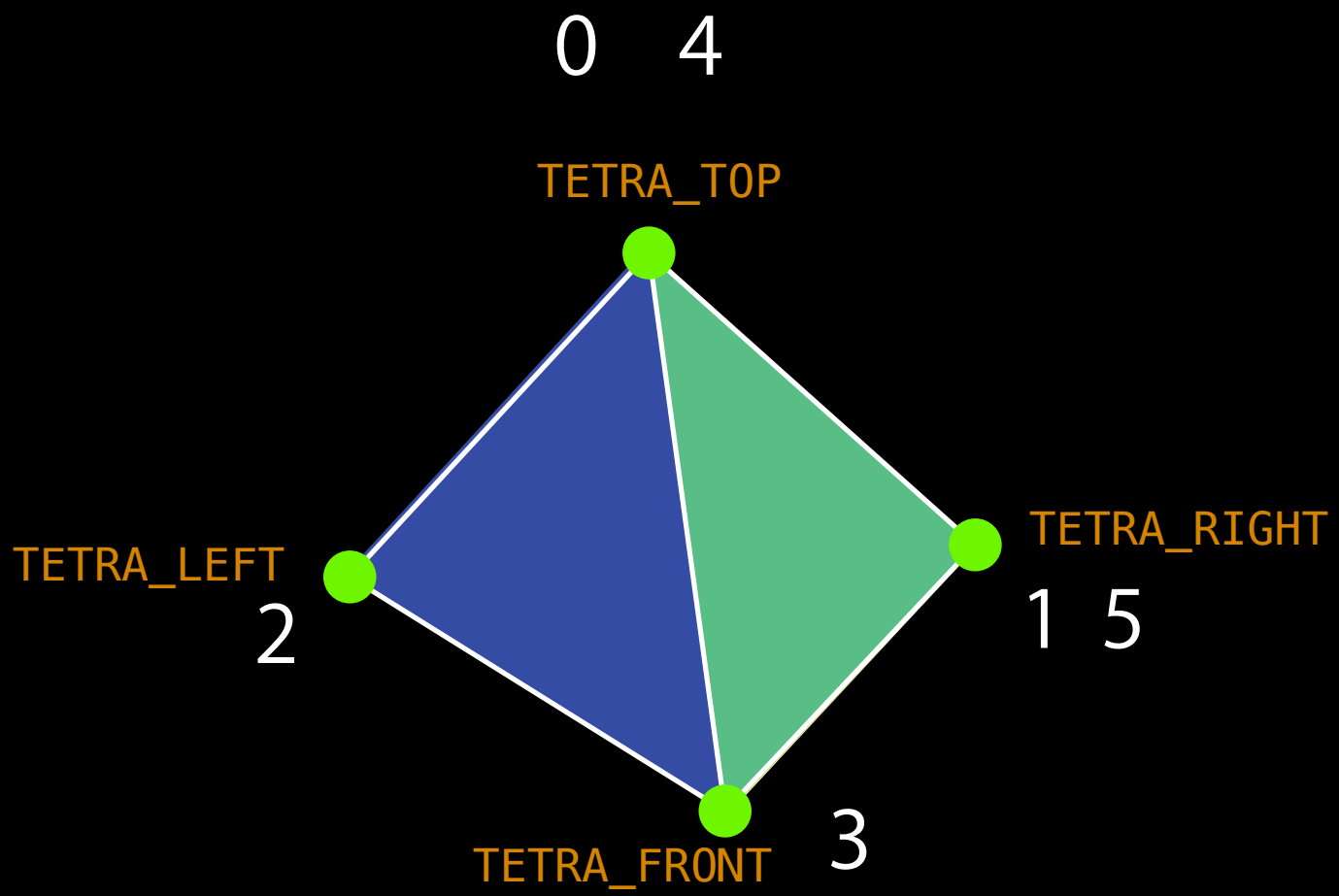
Drawing a Tetrahedron

- Use a triangle strip



Drawing a Tetrahedron

- Use a triangle strip



Demo

Geometry

Geometry Cheat Sheet

```
GLfloat vertexArray[] = {
    x1,y1,z1,
    x2,y2,z2,
    x3,y3,z3, ... };

GLubyte colorArray[] = {
    r1,g1,b1,a1,
    r2,g2,b2,a2,
    r3,g3,b3,a3, ... };

glVertexPointer(dimOfVertices, GL_FLOAT, arrayOffset, vertexArray);
glEnableClientState(GL_VERTEX_ARRAY);
glColorPointer(4, GL_UNSIGNED_BYTE, arrayOffset, colorArray);
glEnableClientState(GL_COLOR_ARRAY);

glDrawArrays(GL_TRIANGLE_STRIP, arrayOffset, numberOfVertices);
   // or GL_TRIANGLE_FAN, GL_TRIANGLES

glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
```

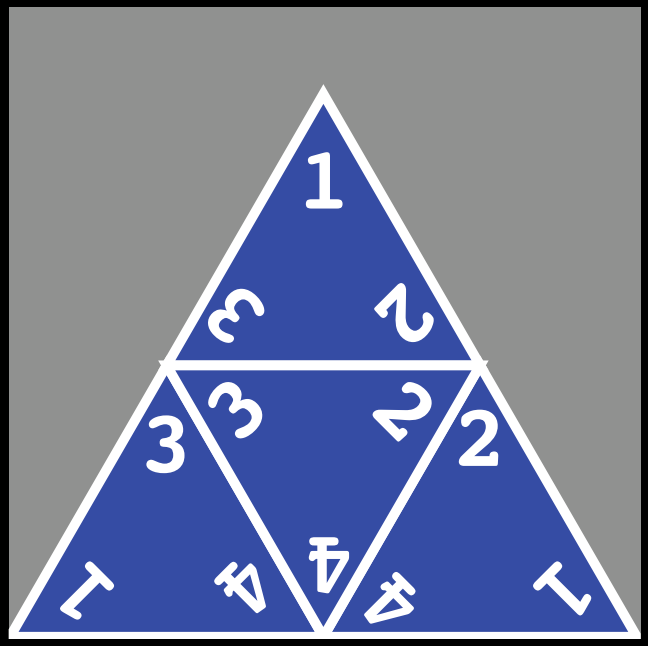
Using Textures

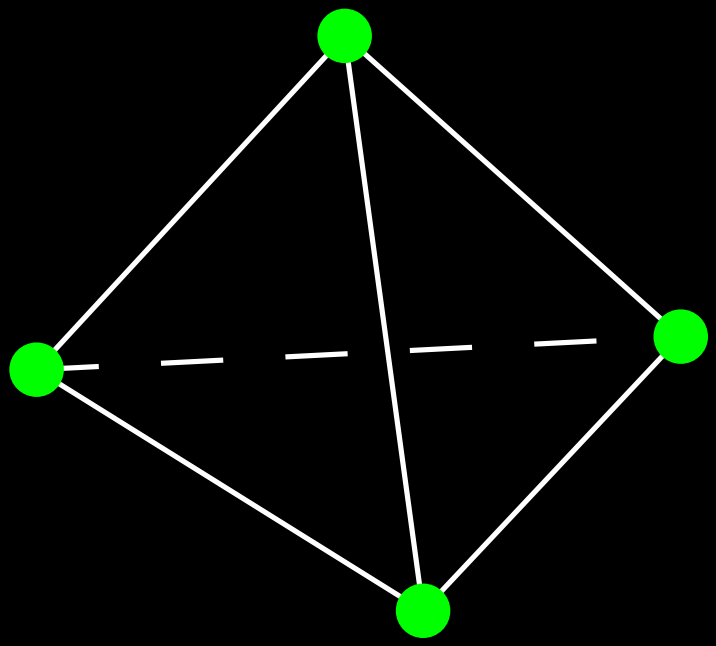
Texture Mapping

- Color pixels according to an image in memory
- Almost always 2D (3D textures are possible though)
- Vertices are given texture coordinates (u,v)

Texture Atlasing

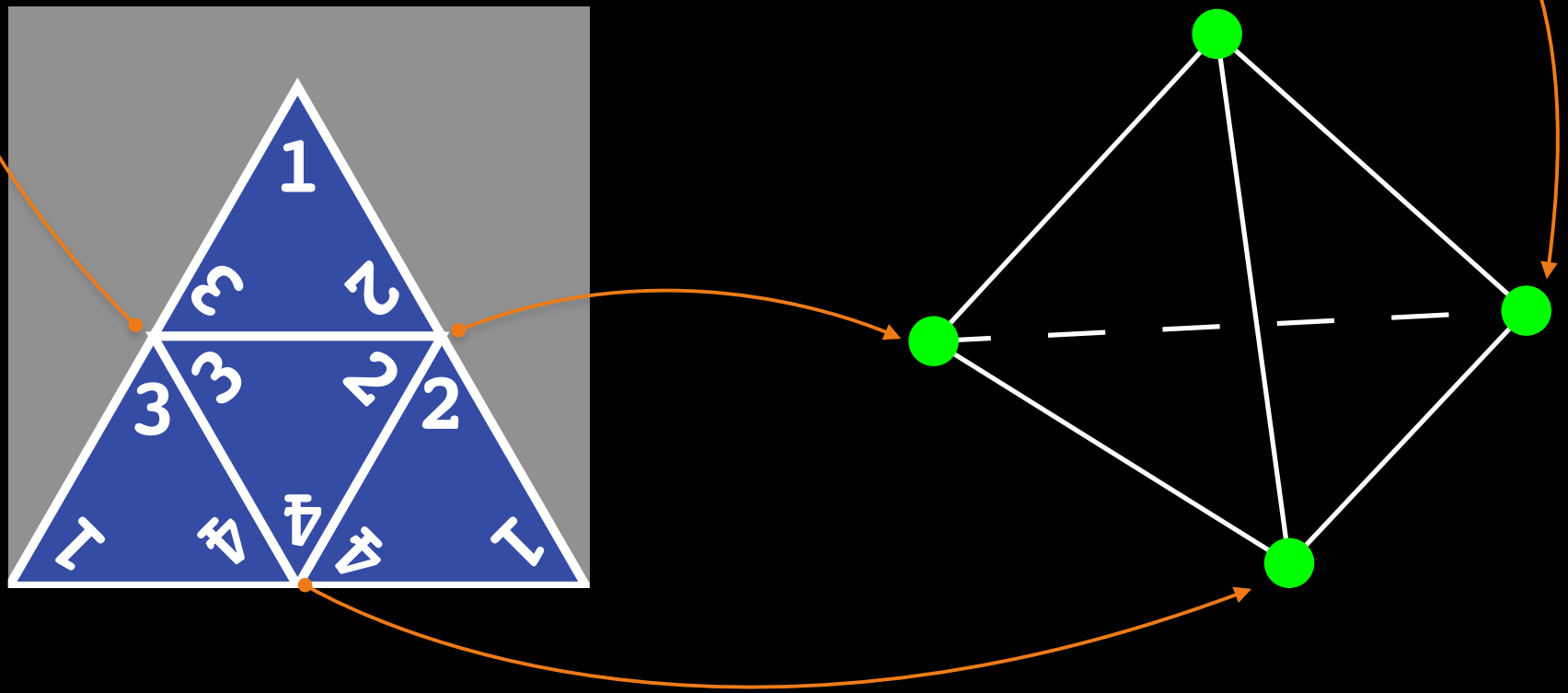
- Swapping textures often is inefficient
- Instead make one giant shared texture





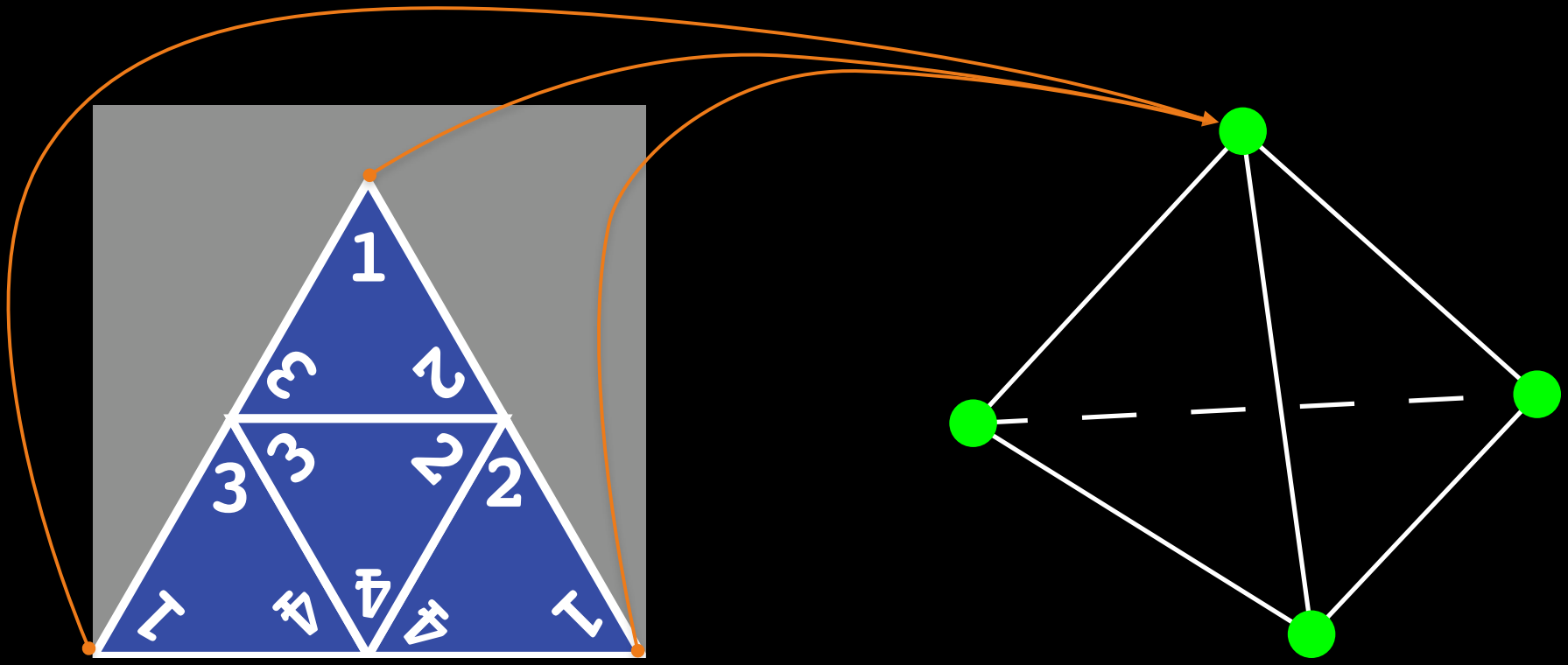
Texture Atlasing

- Swapping textures often is inefficient
- Instead make one giant shared texture



Texture Atlasing

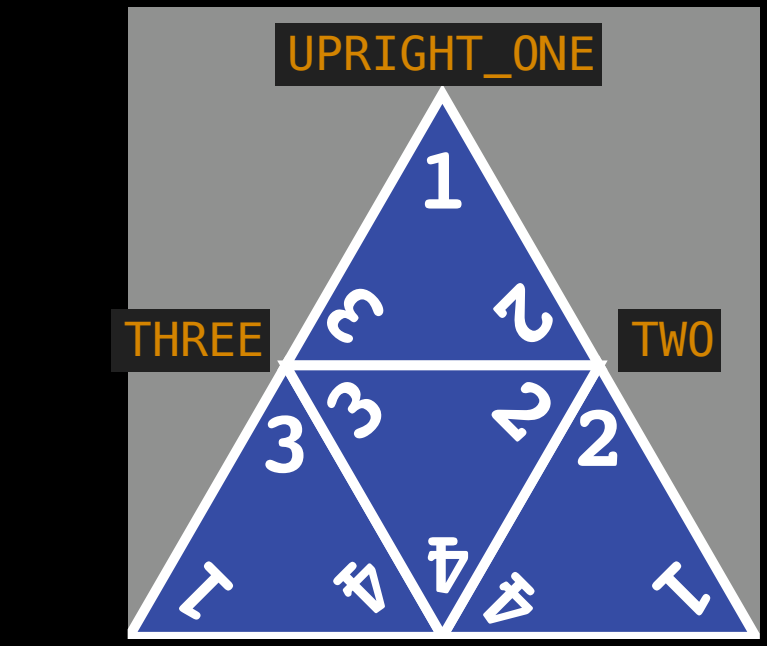
- Swapping textures often is inefficient
- Instead make one giant shared texture

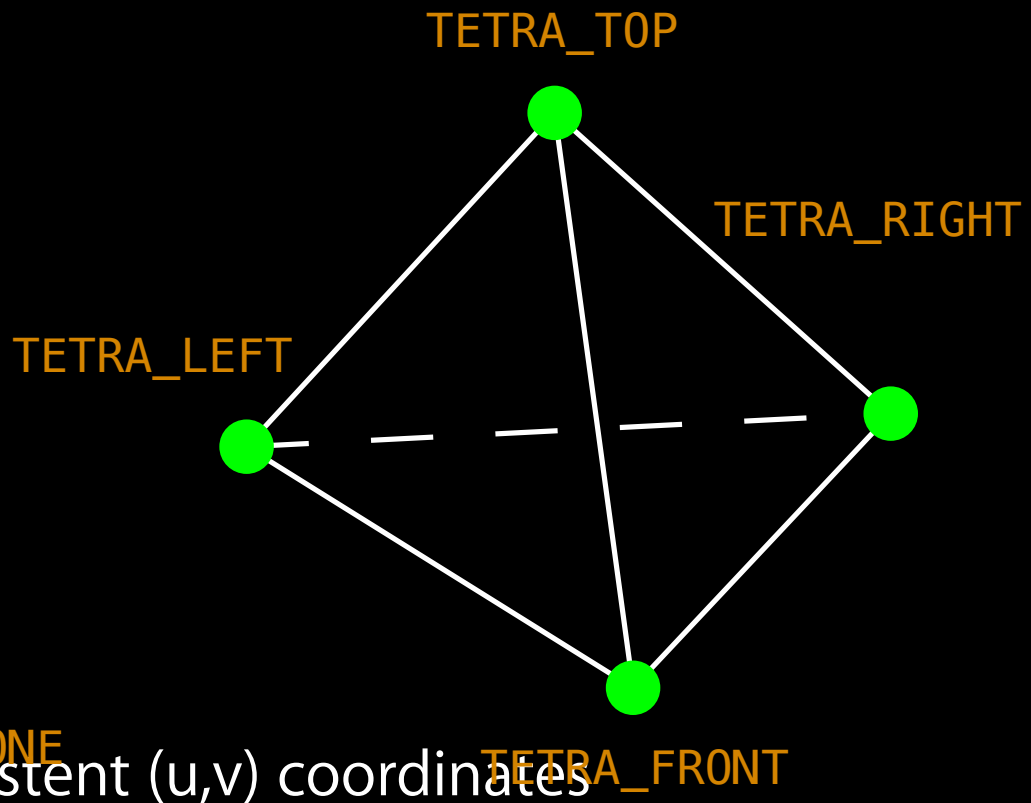


- Vertices may not have consistent (u,v) coordinates

Texture Atlasing

- Swapping textures often is inefficient
- Instead make one giant shared texture





- Vertices may not have consistent (u,v) coordinates

Demo

Textures

Texture Cheat Sheet

```
bash$ export PATH=${PATH}:/Developer/Platforms/iPhoneOS.platform/Developer/usr/bin/
bash$ texturetool -f PVR -e PVRTC image.png -o image.pvrtc
bash$ # image.png must be square with power of side length -- e.g. 64, 256, 1024

#import "PVRTexture.h"
// From Apple's PVRTextureLoader Example Project

NSString * path = [[NSBundle mainBundle] pathForResource: @"image" ofType:@"pvrtc"];
PVRTexture * texture = [[PVRTexture alloc] initWithContentsOfFile: path];

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 1.0f);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture.name);

GLFloat textureCoordArray[] = {
    u1,v1,
    u2,v2,
    u3,v3, ... };

glTexCoordPointer(2, GL_FLOAT, arrayOffset, textureCoordArray);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glDrawArrays(GL_TRIANGLE_STRIP, arrayOffset, numberOfVertices);
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
```

Other Details

OpenGL ES 1.1 vs. ES 2.0

- This lecture described OpenGL ES1.1
- ES 2.0 is drastically different
  - Uses shader based approach
  - More flexible, harder to wrap your head around
- ES 2.0 not available in iPhones before 3GS

Want to know more?

- Apple OpenGL Programming Guide
- OpenGL Redbook
- The Internets
- Stanford CS 148, 248

- Topics of interest
  - Framebuffers
  - Depth Testing
  - Backface Culling
  - Animation (not inherently part of OpenGL)
  - Transparency and Blending
  - Lighting and Shading

Questions?