Advanced OpenGL ES

iPhone and iPod touch Development
Fall 2009 — Lecture 19

Questions?

Announcements

- Assignment #7 due Wednesday, November 11th by 11:59pm

Today's Topics

- Wiring Up Touches
- Moving Around
- Textures
- Lighting & Materials
- Shaders
- Particle System Demo

A Note About These Examples

- All of the examples (unless otherwise noted) use Jeff LaMarche's OpenGL ES template as the project template
- The bulk of things are setup for you using this template, the two things we really care about are...
- GLViewController's -drawView: method
- GLViewController's -setupView: method

Wiring up Touches

Wiring up Touches

- We've already seen how to support touches, let's wire up a 3D cube so that a user can spin it by dragging their finger around the screen
- To do this we'll measure how much and in what direction the user moved their finger and apply a rotation to the scene to track their movements

GLViewController.h

```
#import <UIKit/UIKit.h>
#import "GLView.h"

@interface GLViewController : UIViewController <GLViewDelegate> {
  float oldX;
  float oldY;
  float dx;
  float dy;
  float angle;
}

@property(nonatomic, assign) float oldX;
@property(nonatomic, assign) float oldY;
@property(nonatomic, assign) float dx;
@property(nonatomic, assign) float dy;
@property(nonatomic, assign) float angle;

@end
```

GLViewController.m

```
#import "GLViewController.h"
#import "ConstantsAndMacros.h"
#import "OpenGLCommon.h"

#define kScreenWidth  320
#define kScreenHeight 480

@implementation GLViewController
@synthesize oldX;
@synthesize oldY;
@synthesize dx;
@synthesize dy;
@synthesize angle;

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
  CGPoint loc = [[touches anyObject] locationInView:self.view];
  self.oldX = loc.x;
  self.oldY = loc.y;
}

// ...
```

GLViewController.m

```
// ...

- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event {
  CGPoint loc = [[touches anyObject] locationInView:self.view];
  self.dx = loc.x - self.oldX;;
  self.dy = loc.y - self.oldY;
  self.oldX = loc.x;
  self.oldY = loc.y;
  self.angle = 180 * sqrt(self.dx * self.dx + self.dy * self.dy) / kScreenWidth;
  [(id)self.view drawView];
}

// ...
```

GLViewController.m

```
// ...

- (void)drawView:(UIView *)theView {

  glColor4f(0.0, 0.0, 0.0, 0.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  static const GLfloat cubeVertices[] = {
    /* ... same as previous lecture */
  };
  static const GLubyte cubeNumberOfIndices = 36;

  const GLubyte cubeVertexFaces[] = {
    /* ... same as previous lecture */
  };
  const GLubyte cubeFaceColors[] = {
    /* ... same as previous lecture */
  };

  // ...
```

GLViewController.m

```
// ...

/* save current rotation state */
GLfloat matrix[16];
glGetFloatv(GL_MODELVIEW_MATRIX, matrix);

/* re-center cube, apply new rotation */
glLoadIdentity();
glRotatef(self.angle, self.dy,self.dx,0);
  /* reapply other rotations so far */
  glMultMatrixf(matrix);

  glEnableClientState(GL_VERTEX_ARRAY);
  /*
    ...
    actually draw the geometry via glDrawElements as in previous lectures
    ...
  */
  glDisableClientState(GL_VERTEX_ARRAY);

}

// ...
```

GLViewController.m

```
// ...

-(void)setupView:(GLView*)view {
  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  float xMax = 2;
  float yMax = xMax * kScreenHeight / kScreenWidth;
  glOrthof(-xMax, xMax,          /* x extents */
           -yMax, yMax,          /* y extents */
           -xMax, xMax);         /* z extents */
  glViewport(0, 0, kScreenWidth, kScreenHeight);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

- (void)dealloc {
  [super dealloc];
}

@end
```

Removing the Status Bar

- Let's remove the status bar at the top of the screen so the user gets a more immersive experience
- To do this we need to add the highlighted entry to the plist...

Key	Value
Information Property List	(13 items)
Status bar is initially hidden	☑
Localization native development re	English
Bundle display name	${PRODUCT_NAME}
Executable file	${EXECUTABLE_NAME}
Icon file	
Bundle identifier	com.yourcompany.${PRODUCT_NAME:rfc1034identifier
InfoDictionary version	6.0
Bundle name	${PRODUCT_NAME}
Bundle OS Type code	APPL
Bundle creator OS Type code	????
Bundle version	1.0
Application requires iPhone enviror	☑
Main nib file base name	MainWindow

- Or, the XML if you prefer...

```
<key>UIStatusBarHidden</key>
<true/>
```

The Resulting App

Moving Around

Moving Around

- Let's say we want to display a small scene and have the user be able to navigate around it — how would we do that?
- Let's support 4 operations — move forwards, move backwards, turn left & turn right
- These operations would move a virtual camera around in a 3D world
- Wouldn't it be nice if there was a function that allowed us to do that without having to perform all the underlying matrix math

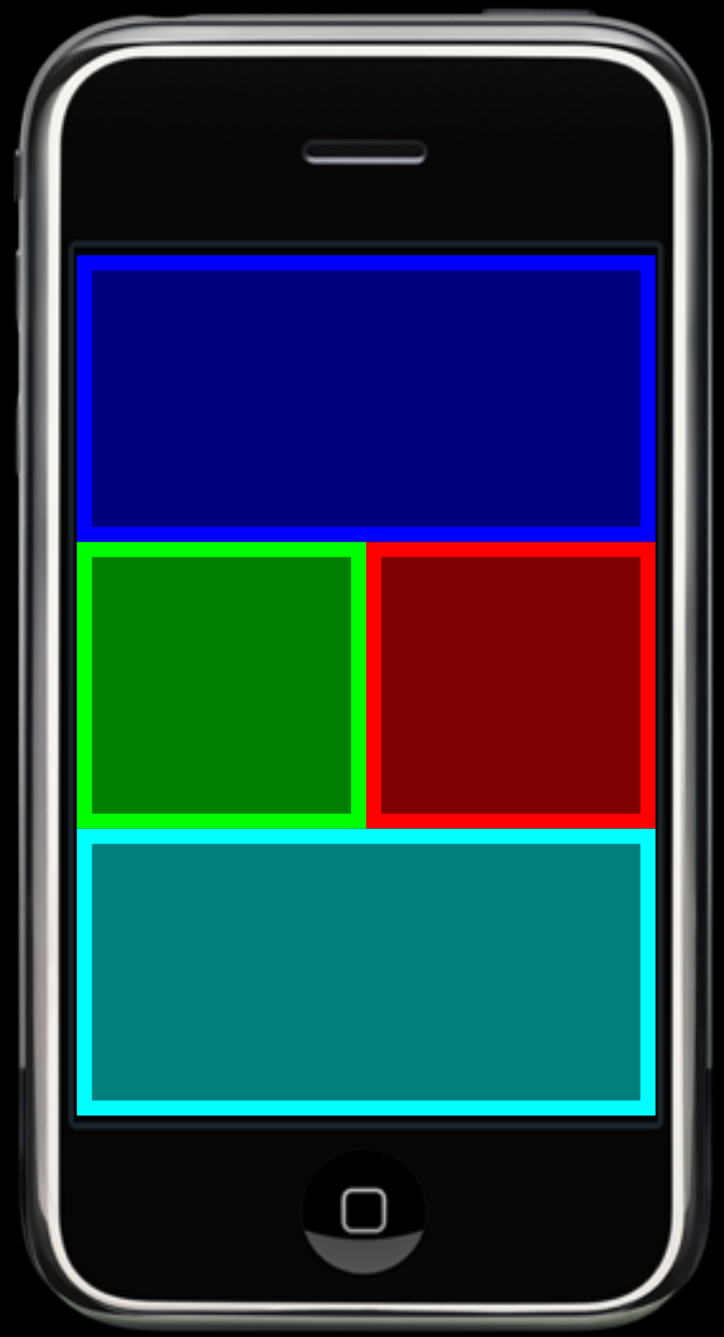
gluLookAt

- There's actually a commonly used function to do just that called gluLookAt
- However this is part of the OpenGL Utility Library and not "core" GL, as such it's not available on the iPhone OS
- However, this code can be obtained from the Mesa 3D project (a software-backed OpenGL implementation)
  - http://mesa3d.org/
- This code is actually already in the OpenGL ES template provided by Jeff LaMarche, the signature is as follows...

```
void gluLookAt(GLfloat eyex, GLfloat eyey, GLfloat eyez,
               GLfloat centerx, GLfloat centery, GLfloat centerz,
               GLfloat upx, GLfloat upy, GLfloat upz);
```

Control Scheme

- If the user touches in the top 160px, then move go forwards
- Else, if the user touches in the bottom 160px, then go backwards
- Else, if the user touches in the remaining left half, turn left
- Else, the user must be touching the remaining section to turn right



Control scheme and associated code in this example based on tutorial by Simon Maurice (see additional resources slide for link)

GLViewController.h

```
#import <UIKit/UIKit.h>
#import "GLView.h"

typedef enum {
  ActionNone,
  ActionMoveForward,
  ActionMoveBackward,
  ActionTurnLeft,
  ActionTurnRight
} Action;

@interface GLViewController : UIViewController <GLViewDelegate> {
  Action action;
  GLfloat center[3];
  GLfloat eye[3];
}

@end
```

GLViewController.m

```
#import "GLViewController.h"
#import "ConstantsAndMacros.h"
#import "OpenGLCommon.h"

#define SPEED_MOVE 0.025
#define SPEED_TURN 0.05

@implementation GLViewController

- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    CGPoint pos = [[touches anyObject] locationInView:self.view];
    if (pos.y < 160) {
        action = ActionMoveForward;
    } else if (pos.y > 320) {
        action = ActionMoveBackward;
    } else if (pos.x < 160) {
        action = ActionTurnLeft;
    } else {
        action = ActionTurnRight;
    }
}
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event {
    action = ActionNone;
}

// ...
```

GLViewController.m

```
// ...
- (void)handleTouches {
  if (action != ActionNone) {
    GLfloat v[] = {center[0] - eye[0], center[1] - eye[1], center[2] - eye[2]};
    switch (action) {
      case ActionMoveForward:
        eye[0] += v[0] * SPEED_MOVE; eye[2] += v[2] * SPEED_MOVE;
        center[0] += v[0] * SPEED_MOVE; center[2] += v[2] * SPEED_MOVE;
        break;
      case ActionMoveBackward:
        eye[0] -= v[0] * SPEED_MOVE; eye[2] -= v[2] * SPEED_MOVE;
        center[0] -= v[0] * SPEED_MOVE; center[2] -= v[2] * SPEED_MOVE;
        break;
      case ActionTurnLeft:
        center[0] = eye[0] + cos(-SPEED_TURN)*v[0] - sin(-SPEED_TURN)*v[2];
        center[2] = eye[2] + sin(-SPEED_TURN)*v[0] + cos(-SPEED_TURN)*v[2];
        break;
      case ActionTurnRight:
        center[0] = eye[0] + cos(SPEED_TURN)*v[0] - sin(SPEED_TURN)*v[2];
        center[2] = eye[2] + sin(SPEED_TURN)*v[0] + cos(SPEED_TURN)*v[2];
        break;
    }
  }
}
// ...
```

GLViewController.m

```
// ...

- (void)drawView:(UIView *)theView {

  // floor
  const GLfloat quadVertices[] = {
    -10.0f,  0.0f, 10.0,
     10.0f,  0.0f, 10.0,
     10.0f, 0.0f, -10.0,
    -10.0f, 0.0f, -10.0
  };

  // cube
  static const GLfloat cubeVertices[] = {
    /* ... same as earlier examples ... */
  };
  static const GLubyte cubeNumberOfIndices = 36;
  const GLubyte cubeVertexFaces[] = {
    /* ... same as earlier examples ... */
  };
  const GLubyte cubeFaceColors[] = {
    /* ... same as earlier examples ... */
  };

  // ...
```

GLViewController.m

```
// ...

// GL setup
glColor4f(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glEnableClientState(GL_VERTEX_ARRAY);

// setup the view
glLoadIdentity();
[self handleTouches];
gluLookAt(eye[0], eye[1], eye[2],
          center[0], center[1], center[2],
          0.0, 1.0, 0.0);

// draw the floor
glColor4f(.25, .25, .75, 1);
glVertexPointer(3, GL_FLOAT, 0, quadVertices);
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

// ...
```

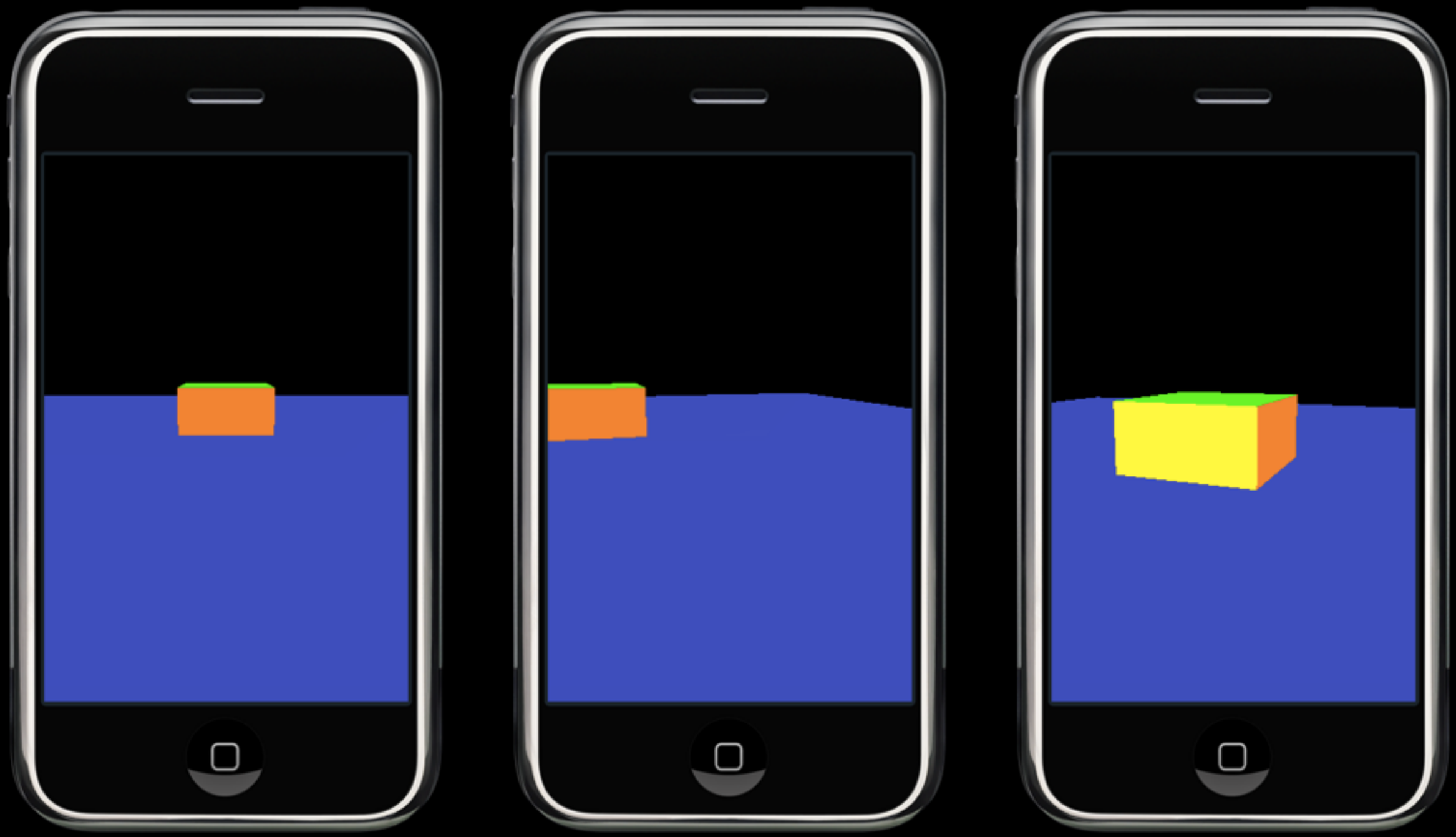
GLViewController.m

```
// ...

// draw a cube on the floor
glVertexPointer(3, GL_FLOAT, 0, cubeVertices);
int colorIndex = 0;
for(int i = 0; i < cubeNumberOfIndices; i += 3) {
  glColor4ub(cubeFaceColors[colorIndex], cubeFaceColors[colorIndex+1],
             cubeFaceColors[colorIndex+2], cubeFaceColors[colorIndex+3]);
  int face = (i / 3.0);
  if (face % 2 != 0.0) {
    colorIndex += 4;
  }
  glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_BYTE, &cubeVertexFaces[i]);
}

// GL teardown
glDisableClientState(GL_VERTEX_ARRAY);

}

// ...
```

GLViewController.m

```
// ...

-(void)setupView:(GLView*)view {
  const GLfloat zNear = 0.1, zFar = 100.0, fieldOfView = 45.0;
  GLfloat size = zNear * tanf(DEGREES_TO_RADIANS(fieldOfView) / 2.0);

  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_PROJECTION);
  CGRect rect = self.view.bounds;
  glFrustumf(-size, size, -size / (rect.size.width / rect.size.height),
             size / (rect.size.width / rect.size.height), zNear, zFar);
  glViewport(0, 0, rect.size.width, rect.size.height);
  glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

  eye[0] = -10; eye[1] = 3; eye[2] = -10;
  center[0] = 0; center[1] = 0; center[2] = 0;

  glMatrixMode(GL_MODELVIEW);
}

// ...

@end
```

The Resulting App



Textures

Textures

- OpenGL ES supports mapping textures (images) onto surfaces
- Images must be square and the width and hight must be a power of 2
- Basic steps
  - Load image into memory from disk
  - Copy image into GL subsystem
  - Define texture mapping
  - Draw

OpenGL Texture Coordinates

- Similar to specifying colors for each vertex, but instead we define a coordinate in the texture
- We'll define these in an array that coincides with the order we've declared vertices

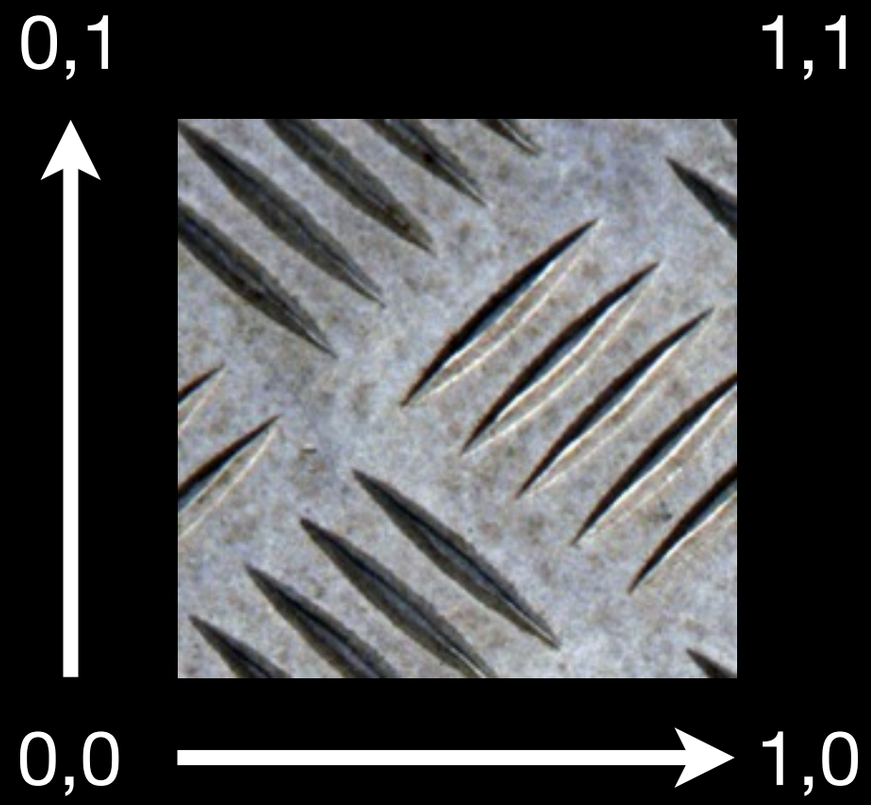


Image & parts of the code in this example based on tutorial by Simon Maurice (see additional resources slide for link)

GLViewController.h

```
#import <UIKit/UIKit.h>
#import "GLView.h"

@interface GLViewController : UIViewController <GLViewDelegate> {

    // To store the textures (we only have 1)
    GLuint textures[1];

}
@end
```

GLViewController.m

```
#import "GLViewController.h"
#import "ConstantsAndMacros.h"
#import "OpenGLCommon.h"

@implementation GLViewController

- (void)drawView:(UIView *)theView {

  static GLfloat rquad;

  const GLfloat quadVertices[] = {
    -1.0f,  1.0f, 0.0f,
     1.0f,  1.0f, 0.0f,
     1.0f, -1.0f, 0.0f,
    -1.0f, -1.0f, 0.0f
  };

  const GLshort squareTextureCoords[] = {
    0, 1,       // top left
    0, 0,       // bottom left
    1, 0,       // bottom right
    1, 1        // top right
  };

  // ...
```

GLViewController.m

```
// ...

// setup GL
glClear(GL_COLOR_BUFFER_BIT);
glLoadIdentity();
glTranslatef(0, 0, -3);

glColor4f(1.0, 1.0, 1.0, 1.0);

// tell GL about our texture coordinates
glTexCoordPointer(2, GL_SHORT, 0, squareTextureCoords);

// turn on texture array support (NEW)
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);

// draw square
glRotatef(rquad, 1.0f, 0.0f, 0.0f);
glVertexPointer(3, GL_FLOAT, 0, quadVertices);
glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

// tear-down GL
glDisableClientState(GL_TEXTURE_COORD_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);

// ...
```

GLViewController.m

```
// ...

// update rotation
static NSTimeInterval lastDrawTime;
if (lastDrawTime) {

  NSTimeInterval timeSinceLastDraw = [NSDate timeIntervalSinceReferenceDate] -
                                     lastDrawTime;
  rquad -= 40.0 * timeSinceLastDraw;

    if (rquad < 0.0) {
      rquad += 360.0;
    }

  }

  lastDrawTime = [NSDate timeIntervalSinceReferenceDate];

}

// ...
```

GLViewController.m

```
- (void)loadTextures {
  // load image as a CG ref
  CGImageRef textureImage = [UIImage imageNamed:@"texture.png"].CGImage;

  // if failed, bail
  if (!textureImage) {
    NSLog(@"Error: failed to load texture");
    return;
  }

  // figure out the width and height
  int texWidth = CGImageGetWidth(textureImage);
  int texHeight = CGImageGetHeight(textureImage);

  // alloc space for the texture
  GLubyte *textureData = (GLubyte *)malloc(texWidth * texHeight * 4);

  // create a CA context ref
  CGContextRef textureContext = CGBitmapContextCreate(
    textureData, texWidth, texHeight, 8, texWidth * 4,
    CGImageGetColorSpace(textureImage),
    kCGImageAlphaPremultipliedLast
  );

  // ...
```

GLViewController.m

```
// ...

// draw the image to in-memory buffer
CGContextDrawImage(textureContext, CGRectMake(0,0,texWidth,texHeight),
                   textureImage);

// done with context - release it
CGContextRelease(textureContext);

// have GL create handle for our texture
glGenTextures(1, &textures[0]);

// tell GL that the image is 2D
glBindTexture(GL_TEXTURE_2D, textures[0]);

// send our data down to GL, copy into graphics hardware
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, texWidth, texHeight, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, textureData);

// free our in-memory copy of the data
free(textureData);

// ...
```

GLViewController.m

```
    // ...

    // specify min/max filters
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // tell GL to turn on textures
    glEnable(GL_TEXTURE_2D);
}

// ...
```

GLViewController.m

```
// ...

-(void)setupView:(GLView*)view {

   const GLfloat zNear = 0.1, zFar = 1000.0, fieldOfView = 60.0;
   GLfloat size = zNear * tanf(DEGREES_TO_RADIANS(fieldOfView) / 2.0);

   glMatrixMode(GL_PROJECTION);
   CGRect rect = view.bounds;
   glFrustumf(-size, size, -size / (rect.size.width / rect.size.height), size /
              (rect.size.width / rect.size.height), zNear, zFar);
   glViewport(0, 0, rect.size.width, rect.size.height);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
   [self loadTextures];

}

// ...

@end
```

The Resulting App

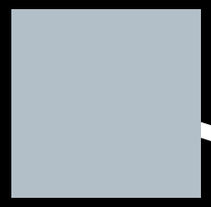
Adjusting the Color

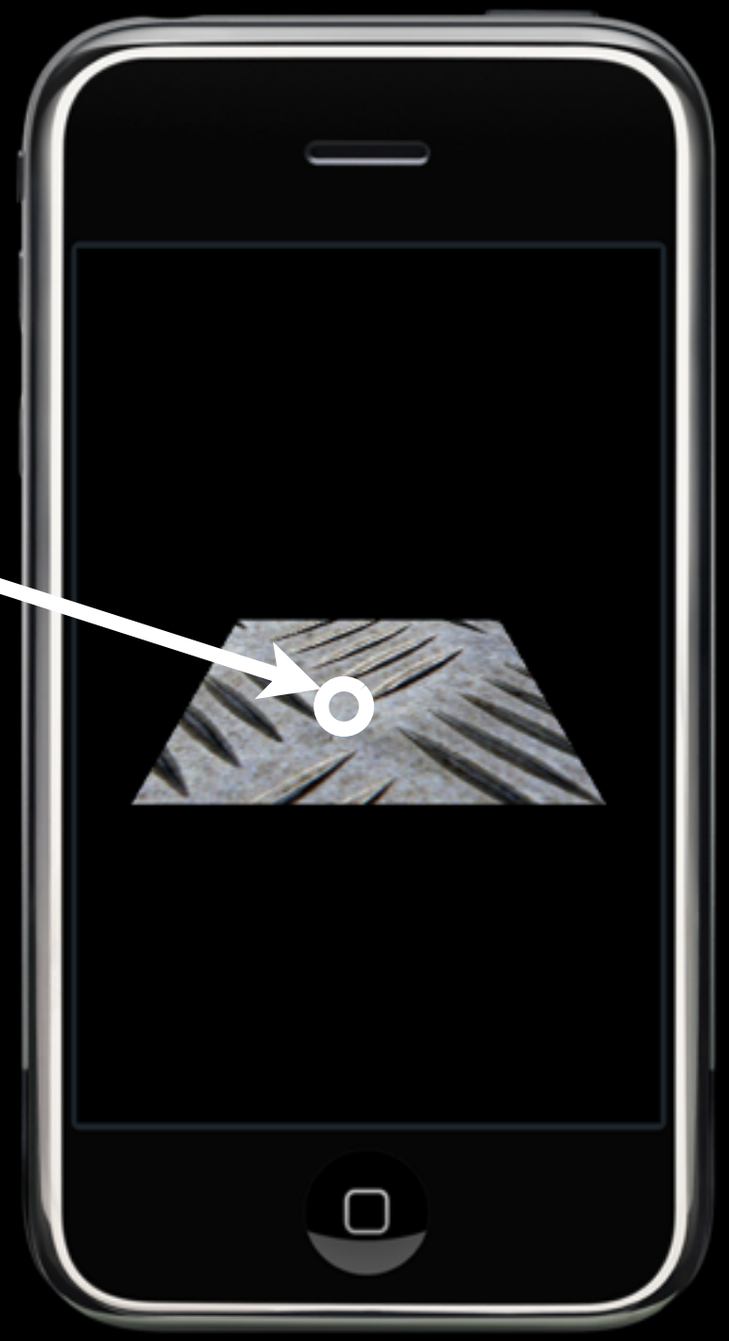
- In the previous example we set the color to be white
- Though not obvious, setting this color effected our rendering
- OpenGL will actually multiply each component value (RGBA) by the color's value to calculate the result
- We can use this to recolor our texture

Adjusting the Color

- So, if we have a color in our texture of the following...

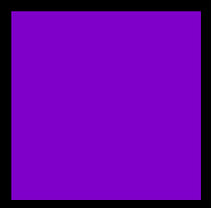
(.70, .75, .78, 1.0)





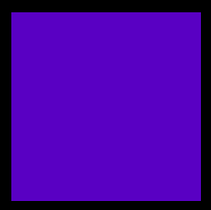
- And we first set the color to...

```
glColor4f(0.5, 0.0, 0.8, 1.0);
```



- We multiply each component's value in the texture with the color's value to get...

(.35, 0.0, .62, 1.0)



The Resulting App

Lighting & Materials

Lights in OpenGL

- OpenGL lights are comprised of three components...
  - The specular component specifies light that is direct & focused — tends to form "hot spots" or shine
  - The diffuse component defines more flat, even directional lighting that shines onto parts of the object facing the light
  - The ambient component is light without a given source — shines light on all sides of all objects

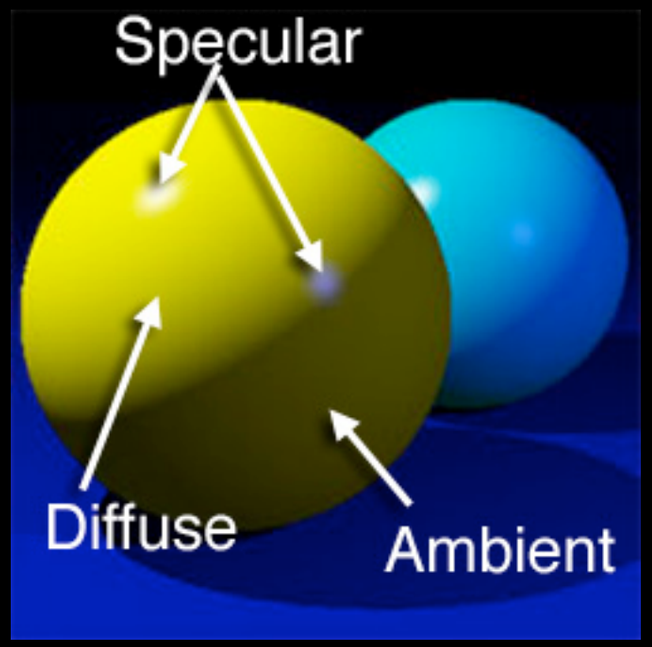


Image & code in this example by Jeff LaMarche (see additional resources slide for link)

Normals

- A surface normal for a flat surface is a vector that is perpendicular to the surface
- This can easily be computed by taking the cross product of the vector formed by 2 adjacent sides of the surface
- Why do we care?
  - Because the normal determines the orientation of the surface towards the light sources in the scene

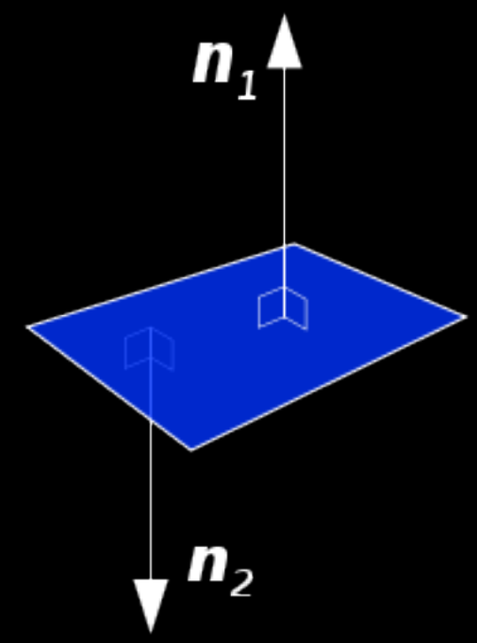


Image from Wikipedia

Enabling Lights in OpenGL

- If we wish to use custom lighting, we need to send OpenGL the command to turn that feature on...

```
glEnable(GL_LIGHTING);
```

- However that statement alone will simply result in a black object, we need to define some lights
- OpenGL ES allows for 8 different lights, OpenGL refers to them as light0 through light7
- To turn on a given light you would issue a command like so...

```
glEnable(GL_LIGHT0);
```

Defining Lighting Values

- Each light component (specular, diffuse & ambient) can be defined in terms of an RGBA value
- So an intense white light might be defined as...

```
GLfloat brightLightComponent[] = {1.0, 1.0, 1.0, 1.0};
```

- Whereas a dim light, might be specified as...

```
GLfloat dimLightComponent[] = {0.2, 0.2, 0.2, 1.0};
```

- You can even color the lights, by differing the values...

```
GLfloat blueLightComponent[] = {0.2, 0.2, 1.0, 1.0};
```

Materials

- OpenGL allows you to define the properties of your surfaces by indicating how your object interacts with light sources...
  - Ambient
  - Diffuse
  - Specular
  - Shininess
  - Emissions

Spheres

- To demonstrate OpenGL lighting, we're going to actually break away from our cube and pyramid geometry to something more interesting — a sphere
- Spheres are frequently created using the gluSphere function, however this is part of the OpenGL Utility Library and not "core" GL, as such it's not available on the iPhone OS
- There are several ways to create spheres out of triangles, we'll use Jeff LaMarche's sphere code for the example that follow

GLViewController.h

```
#import <UIKit/UIKit.h>
#import <OpenGLES/EAGL.h>
#import <OpenGLES/ES1/gl.h>
#import <OpenGLES/ES1/glext.h>
#import "OpenGLCommon.h"

@class GLView;
@interface GLViewController : UIViewController {
  Vertex3D    *sphereTriangleStripVertices;
  Vector3D    *sphereTriangleStripNormals;
  GLuint      sphereTriangleStripVertexCount;

  Vertex3D    *sphereTriangleFanVertices;
  Vector3D    *sphereTriangleFanNormals;
  GLuint      sphereTriangleFanVertexCount;
}
- (void)drawView:(GLView*)view;
- (void)setupView:(GLView*)view;

@end
```

GLViewController.m

```
#import "GLViewController.h"
#import "GLView.h"
#import "ConstantsAndMacros.h"

void getSolidSphere(Vertex3D **triangleStripVertexHandle,
                    Vector3D **triangleStripNormalHandle,
                    GLuint *triangleStripVertexCount,
                    Vertex3D **triangleFanVertexHandle,
                    Vector3D **triangleFanNormalHandle,
                    GLuint *triangleFanVertexCount,
                    GLfloat radius,
                    GLuint slices,
                    GLuint stacks) {

    // Jeff LaMarche's code
    // First 6 args are set by this function (contain vertices and counts)
    // Last 3 define the size of the sphere and horizontal and vertical resulution
}

@implementation GLViewController

// ...
```

GLViewController.m

```
// ...

- (void)drawView:(GLView*)view {

  static GLfloat rot = 0.0;

  // setup GL
  glLoadIdentity();
  glTranslatef(0.0f,0.0f,-3.0f);
  glRotatef(rot,1.0f,1.0f,1.0f);
  glClearColor(0.7, 0.7, 0.7, 1.0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  // set material property for sphere
  GLfloat ambientAndDiffuse[] = {0.0, 0.1, 0.7, 1.0};
  glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, ambientAndDiffuse);
  GLfloat specular[] = {1, 1, 1, 1.0};
  glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specular);
  glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 25.0);

  // ...
```

GLViewController.m

```
// ...

// enable vertex and normal data
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);

// draw fan parts of sphere
glVertexPointer(3, GL_FLOAT, 0, sphereTriangleFanVertices);
glNormalPointer(GL_FLOAT, 0, sphereTriangleFanNormals);
glDrawArrays(GL_TRIANGLE_FAN, 0, sphereTriangleFanVertexCount);

// draw strip parts of sphere
glVertexPointer(3, GL_FLOAT, 0, sphereTriangleStripVertices);
glNormalPointer(GL_FLOAT, 0, sphereTriangleStripNormals);
glDrawArrays(GL_TRIANGLE_STRIP, 0, sphereTriangleStripVertexCount);

// disable vertex and normal data
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_NORMAL_ARRAY);

// ...
```

GLViewController.m

```
// ...

// update rotation
static NSTimeInterval lastDrawTime;
if (lastDrawTime) {
  NSTimeInterval timeSinceLastDraw = [NSDate timeIntervalSinceReferenceDate] -
                                     lastDrawTime;
    rot += 50 * timeSinceLastDraw;
}
  lastDrawTime = [NSDate timeIntervalSinceReferenceDate];
}

// ...
```

GLViewController.m

```
// ...

-(void)setupView:(GLView*)view {

  const GLfloat zNear = 0.01, zFar = 1000.0, fieldOfView = 45.0;
  GLfloat size = zNear * tanf(DEGREES_TO_RADIANS(fieldOfView) / 2.0);

  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_PROJECTION);
  CGRect rect = view.bounds;
  glFrustumf(-size, size, -size / (rect.size.width / rect.size.height), size /
             (rect.size.width / rect.size.height), zNear, zFar);
  glViewport(0, 0, rect.size.width, rect.size.height);
  glMatrixMode(GL_MODELVIEW);
  glShadeModel(GL_SMOOTH);

  // ...
```

GLViewController.m

```
// ...

// Enable lighting
glEnable(GL_LIGHTING);

// Turn the first light on
glEnable(GL_LIGHT0);

// Define the ambient component of the first light
static const Color3D light0Ambient[] = {{0.2, 0.2, 0.2, 1.0}};
glLightfv(GL_LIGHT0, GL_AMBIENT, (const GLfloat *)light0Ambient);

// Define the diffuse component of the first light
static const Color3D light0Diffuse[] = {{0.8, 0.8, 0.8, 1.0}};
glLightfv(GL_LIGHT0, GL_DIFFUSE, (const GLfloat *)light0Diffuse);

// Define the specular component and shininess of the first light
static const Color3D light0Specular[] = {{0.8, 0.8, 0.8, 1.0}};
glLightfv(GL_LIGHT0, GL_SPECULAR, (const GLfloat *)light0Specular);

// ...
```

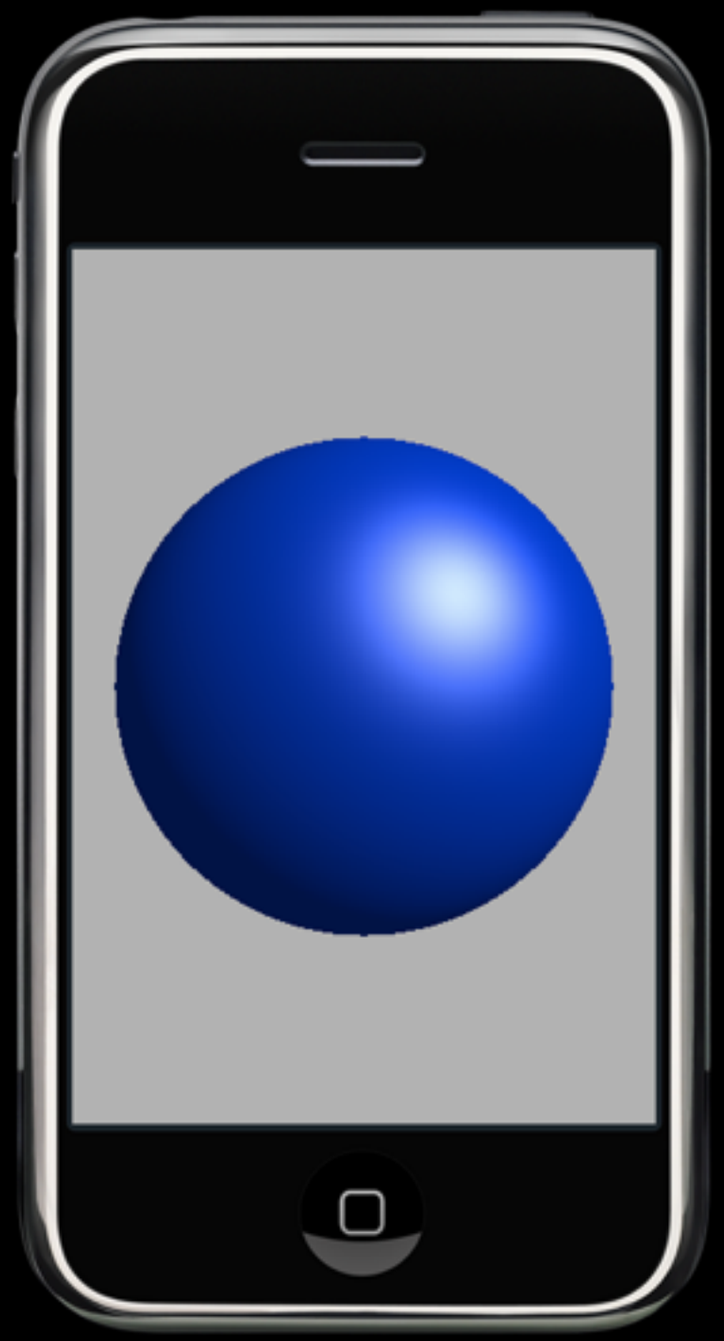
GLViewController.m

```
// ...

// Define the position of the first light
static const Vertex3D light0Position[] = {{5.0, 5.0, 5.0}};
glLightfv(GL_LIGHT0, GL_POSITION, (const GLfloat *)light0Position);

// Calculate light vector so it points at the object
static const Vertex3D objectPoint[] = {{0.0, 0.0, -3.0}};
const Vertex3D lightVector = Vector3DMakeWithStartAndEndPoints(light0Position[0],
                                                               objectPoint[0]);
    glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, (GLfloat *)&lightVector);

    // Define a cutoff angle
    glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 25.0);

    glLoadIdentity();
    glClearColor(0.0f, 0.0f, 0.0f, 1.0f);

    // build sphere geometry
    getSolidSphere(&sphereTriangleStripVertices, &sphereTriangleStripNormals,
                   &sphereTriangleStripVertexCount, &sphereTriangleFanVertices,
                   &sphereTriangleFanNormals, &sphereTriangleFanVertexCount,
                   1.0, 100, 100);
}

// ...
```

GLViewController.m

```
// ...

- (void)dealloc {
    if(sphereTriangleStripVertices)
        free(sphereTriangleStripVertices);
    if (sphereTriangleStripNormals)
        free(sphereTriangleStripNormals);

    if (sphereTriangleFanVertices)
        free(sphereTriangleFanVertices);
    if (sphereTriangleFanNormals)
        free(sphereTriangleFanNormals);

    [super dealloc];
}

@end
```

The Resulting App

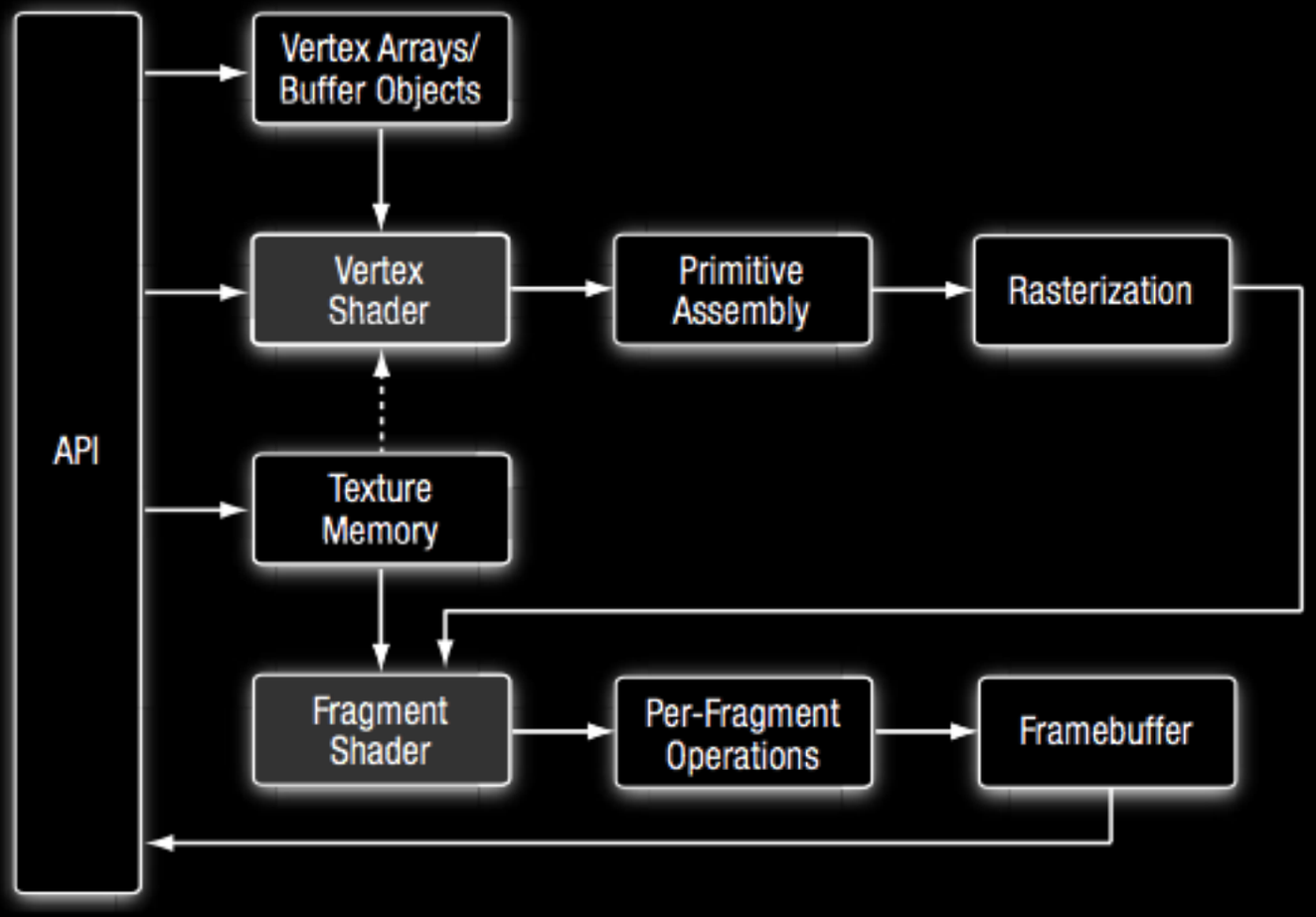


Shading

Shaders

- If we use an OpenGL 2.0 context, you need to write shaders
  - Currently only supported by iPhone 3GS & 3rd generation iPod touch
- Shaders are small programs written in the OpenGL Shader Language (GLSL) that run on the GPU
- The OpenGL "Orange Book" discusses the OpenGL shader language in quite some detail...
  - http://www.3dshaders.com/home/

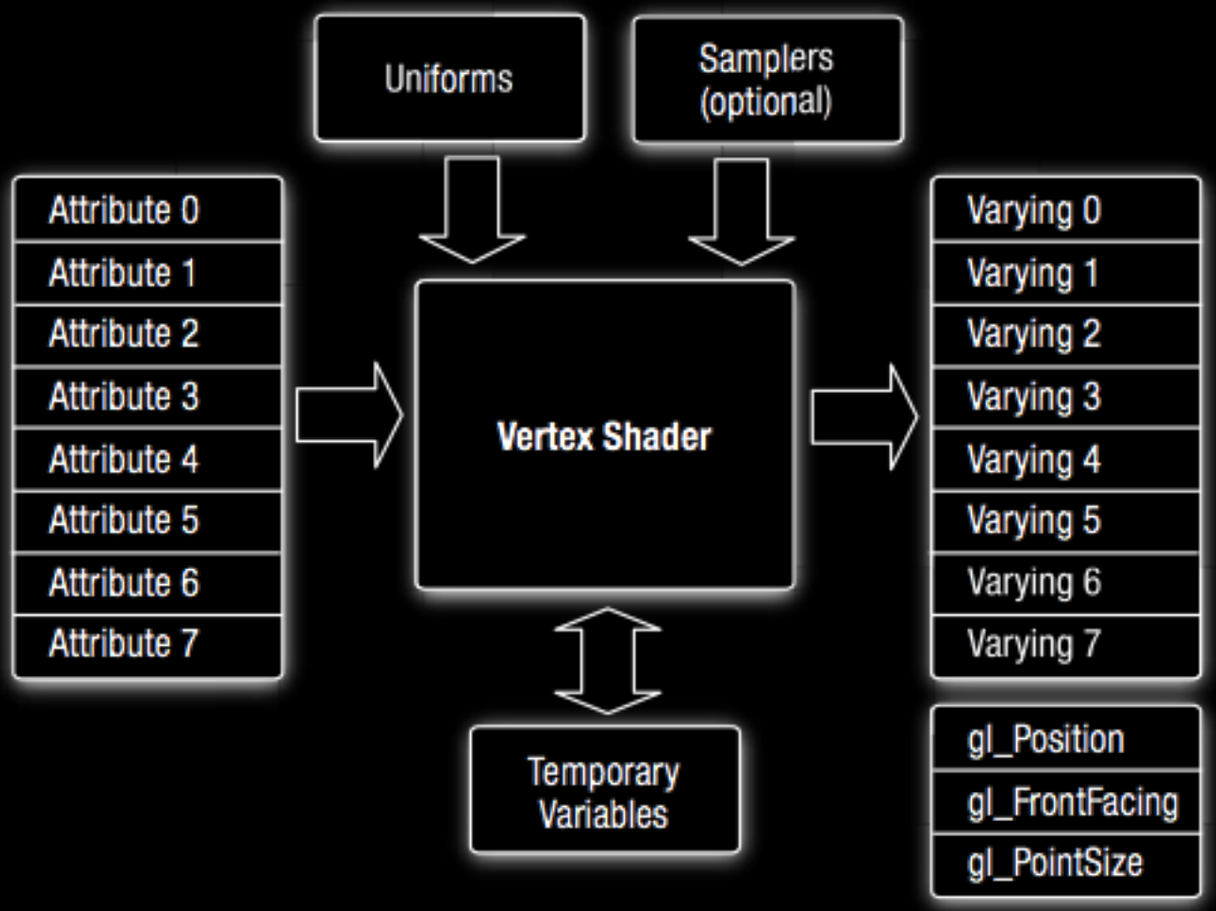
OpenGL ES 2.0 Graphics Pipeline



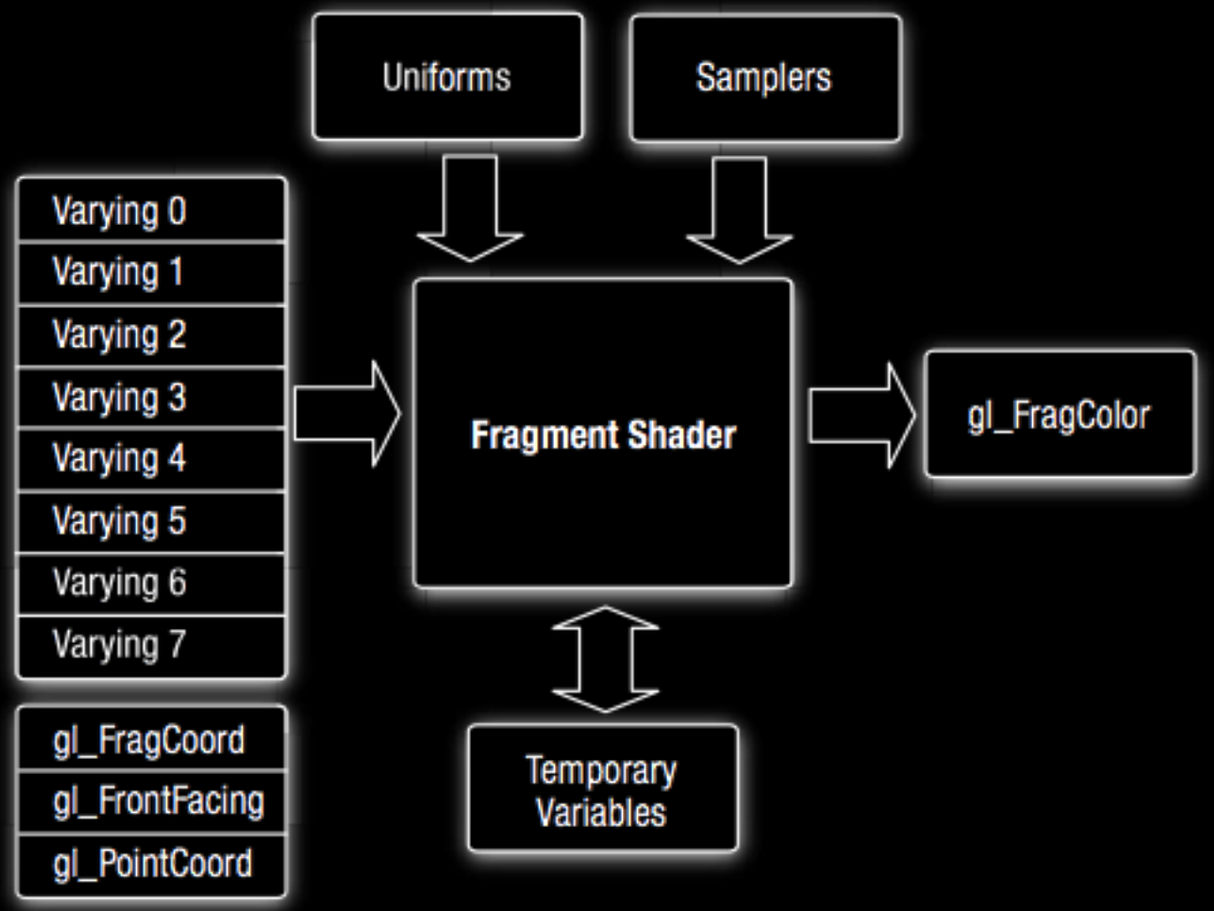
Types of Shaders

- Vertex Shaders
  - Executed for each vertex in shape
  - Outputs gl_Position which can be used to perturb vertices without changing the underlying geometry
  - Can also set other values (i.e. color) via varying outputs
- Fragment Shaders
  - Usually executed for each pixel surface consumes
  - Used to compute color for current position
  - Inputs gl_Position & varying outputs from vertex shader
  - Outputs gl_FragColor to represent pixel color

OpenGL ES 2.0 Vertex Shader



OpenGL ES 2.0 Fragment Shader



Using Shaders

- General process for using shaders...
  - Compile shaders
    - Can be done at runtime or ahead of time
  - Set inputs
  - Tell open GL to use shader (put on GPU)
  - Draw

This example uses Apple's built in OpenGL ES project template

Creating an OpenGL ES 2.0 Rendering Context

- The way you get an OpenGL ES 2.0 context is to try to instantiate and check to see if it is nil
- The following code tries to get a 2.0 context and falls back to a 1.0 context if that that fails...

```
EAGLContext* CreateBestEAGLContext() {
  EAGLContext *context = [[EAGLContext alloc] initWithAPI:kEAGLRenderingAPIOpenGLES2];
  if (!context) {
    context = [[EAGLContext alloc] initWithAPI:kEAGLRenderingAPIOpenGLES1];
  }
  return context;
}
```

- Apple's OpenGL ES Application template does something similar to this

Shader.vsh

```
attribute vec4 position;
attribute vec4 color;

varying vec4 colorVarying;

uniform float translate;

void main()
{
    gl_Position = position;
    gl_Position.y += sin(translate) / 2.0;

    colorVarying = color;
}
```

Shader.fsh

```
varying lowp vec4 colorVarying;

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

Loading Shaders

```
- (BOOL) loadShaders {
  GLuint vertShader, fragShader;
  NSString *vertShaderPathname, *fragShaderPathname;

  // create shader program
  program = glCreateProgram();

  // create and compile vertex shader
  vertShaderPathname = [[NSBundle mainBundle] pathForResource:@"Shader"
                        ofType:@"vsh"];
  if (![self compileShader:&vertShader type:GL_VERTEX_SHADER
        file:vertShaderPathname]) {
    NSLog(@"Failed to compile vertex shader");
    return FALSE;
  }

  // create and compile fragment shader
  fragShaderPathname = [[NSBundle mainBundle] pathForResource:@"Shader"
                        ofType:@"fsh"];
   if (![self compileShader:&fragShader type:GL_FRAGMENT_SHADER
         file:fragShaderPathname]) {
      NSLog(@"Failed to compile fragment shader");
      return FALSE;
   }

  // ...
```

Loading Shaders

```
// ...

// attach vertex shader to program
glAttachShader(program, vertShader);

// attach fragment shader to program
glAttachShader(program, fragShader);

// bind attribute locations
// this needs to be done prior to linking
glBindAttribLocation(program, ATTRIB_VERTEX, "position");
glBindAttribLocation(program, ATTRIB_COLOR, "color");

// link program
if (![self linkProgram:program]) {
  NSLog(@"Failed to link program: %d", program);
  return FALSE;
}

// get uniform locations
uniforms[UNIFORM_TRANSLATE] = glGetUniformLocation(program, "translate");

// ...
```

Loading Shaders

```
// ...

// release vertex and fragment shaders
if (vertShader)
  glDeleteShader(vertShader);
if (fragShader)
  glDeleteShader(fragShader);

return TRUE;
}
```

Compiling a Shader

```
- (BOOL) compileShader:(GLuint *)shader type:(GLenum)type file:(NSString *)file {

  GLint status;
  const GLchar *source;

  source = (GLchar *)[[NSString stringWithContentsOfFile:file
                        encoding:NSUTF8StringEncoding error:nil] UTF8String];
  if (!source) {
    NSLog(@"Failed to load vertex shader");
    return FALSE;
  }

  *shader = glCreateShader(type);
  glShaderSource(*shader, 1, &source, NULL);
  glCompileShader(*shader);

  glGetShaderiv(*shader, GL_COMPILE_STATUS, &status);
  if (status == 0) {
    glDeleteShader(*shader);
    return FALSE;
  }
  return TRUE;

}
```

Rendering

```
- (void) render {

  static const GLfloat squareVertices[] = {
    -0.5f, -0.33f,
     0.5f, -0.33f,
    -0.5f,  0.33f,
     0.5f,  0.33f,
  };

  static const GLubyte squareColors[] = {
    255, 255,   0, 255,
      0, 255, 255, 255,
      0,   0,   0,   0,
    255,   0, 255, 255,
  };

  static float transY = 0.0f;

  // This app only creates a single ctx which is already set current at this point.
  // This call is redundant, but needed if dealing with multiple contexts.
  [EAGLContext setCurrentContext:context];

  // ...
```

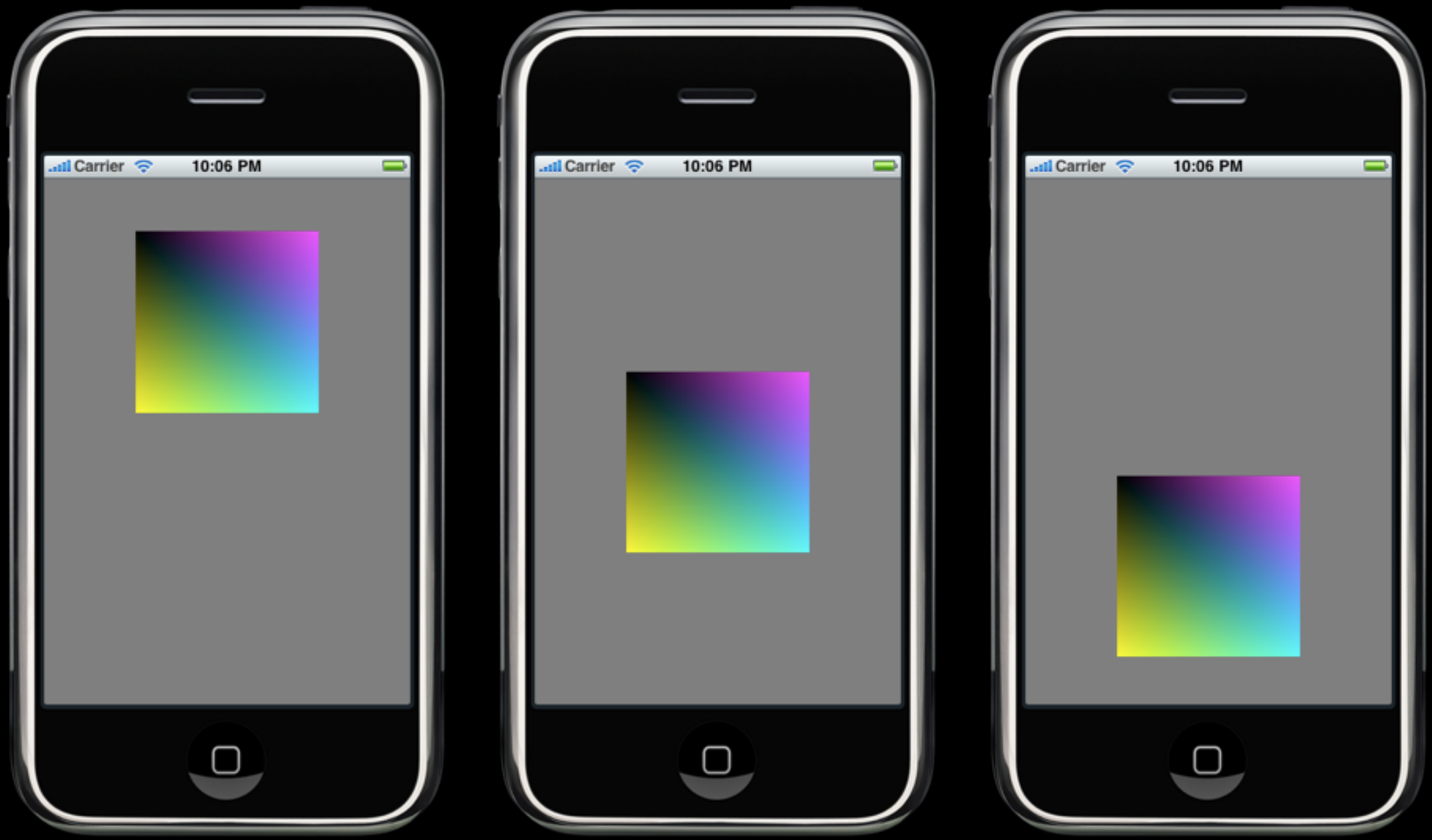
Rendering

```
// ...

// This app creates a default framebuffer which is already bound at this point.
// This call is redundant, but needed if dealing with multiple framebuffers.
glBindFramebuffer(GL_FRAMEBUFFER, defaultFramebuffer);
glViewport(0, 0, backingWidth, backingHeight);

glClearColor(0.5f, 0.5f, 0.5f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT);

// Use shader program
glUseProgram(program);

// Update uniform value
glUniform1f(uniforms[UNIFORM_TRANSLATE], (GLfloat)transY);
transY += 0.075f;

// Update attribute values
glVertexAttribPointer(ATTRIB_VERTEX, 2, GL_FLOAT, 0, 0, squareVertices);
glEnableVertexAttribArray(ATTRIB_VERTEX);
glVertexAttribPointer(ATTRIB_COLOR, 4, GL_UNSIGNED_BYTE, 1, 0, squareColors);
glEnableVertexAttribArray(ATTRIB_COLOR);

// ...
```

Rendering

```
// ...

// Draw
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

// This app creates a single color renderbuffer which is bound at this point.
// This call is redundant, but needed if dealing with multiple renderbuffers.
glBindRenderbuffer(GL_RENDERBUFFER, colorRenderbuffer);
[context presentRenderbuffer:GL_RENDERBUFFER];
}
```

The Resulting App



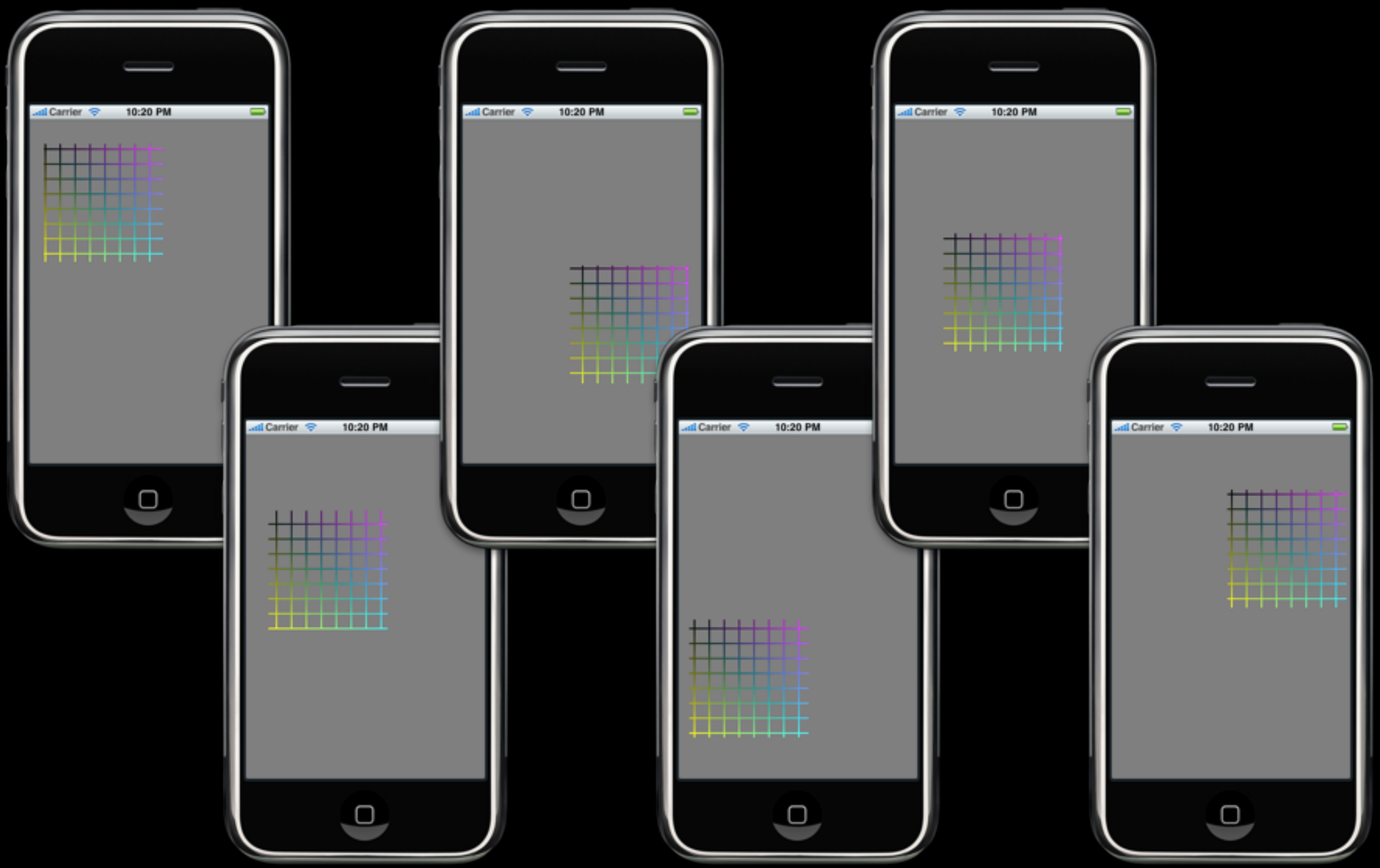
Modified Shader.vsh

```
attribute vec4 position;
attribute vec4 color;

varying vec4 colorVarying;

uniform float translate;

void main()
{
    gl_Position = position;
    gl_Position.x += sin(translate) / 2.0;
    gl_Position.y += sin(translate * .5) / 2.0;

    colorVarying = color;
}
```

Modified Shader.fsh

```
varying lowp vec4 colorVarying;

void main()
{

   if(floor(mod(gl_FragCoord.y, 20.0)) < 2.0 ||
      floor(mod(gl_FragCoord.x, 20.0)) < 2.0 ) {

       gl_FragColor = colorVarying;
   } else {
       discard;
   }
}
```

The Resulting App



Particle System Demo

Particle System Demo

- You can really do some amazing things with OpenGL
- For example, ngmoco:), the company behind games such as Topple and Rolando, open sourced the following particle systems demo...
  - http://gamemakers.ngmoco.com/post/111712416/stanford-university-and-apple-were-kind-enough-to
- If you're really interested in OpenGL ES on the iPhone OS there are also some really useful performance tips in the associated video off that page

Particle System Demo

Additional Resources

- OpenGL ES Programming Guide for iPhone OS
  - http://developer.apple.com/iphone/library/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/
- Khronos Group OpenGL ES pages
  - http://www.khronos.org/opengles/
- OpenGL ES 1.1 online man pages
  - http://www.khronos.org/opengles/sdk/1.1/docs/man/
- OpenGL ES 2.0 online man pages
  - http://www.khronos.org/opengles/sdk/docs/man/

Additional Resources

- Jeff LaMarche's "OpenGL ES from the Ground Up" Series
  - http://iphonedevelopment.blogspot.com/2009/05/opengl-es-from-ground-up-table-of.html
- Simon Maurice's iPhone OpenGL ES Tutorial Series
  - http://web.me.com/smaurice/AppleCoder/iPhone_OpenGL/iPhone_OpenGL.html
- NeHe Production's OpenGL Lessons
  - http://nehe.gamedev.net/lesson.asp?index=01

For Next Class

- Audio Session Programming Guide
  - http://developer.apple.com/iphone/library/documentation/Audio/Conceptual/AudioSessionProgrammingGuide/
- Core Audio Overview
  - http://developer.apple.com/iphone/library/documentation/MusicAudio/Conceptual/CoreAudioOverview/
- Audio Queue Services Programming Guide
  - http://developer.apple.com/iphone/library/documentation/MusicAudio/Conceptual/AudioQueueProgrammingGuide/
- Audio & Video Coding How-To's
  - https://developer.apple.com/iphone/library/codinghowtos/AudioAndVideo/