Texture Mapping, Blending

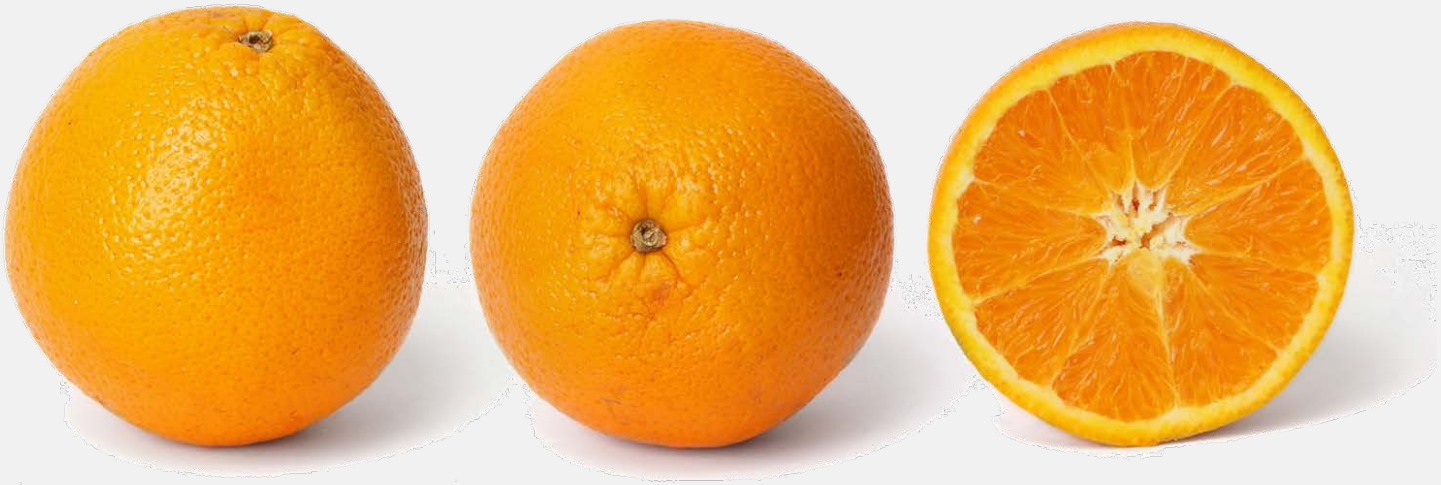
Junho Kim

Kookmin University

Texture Mapping Basics

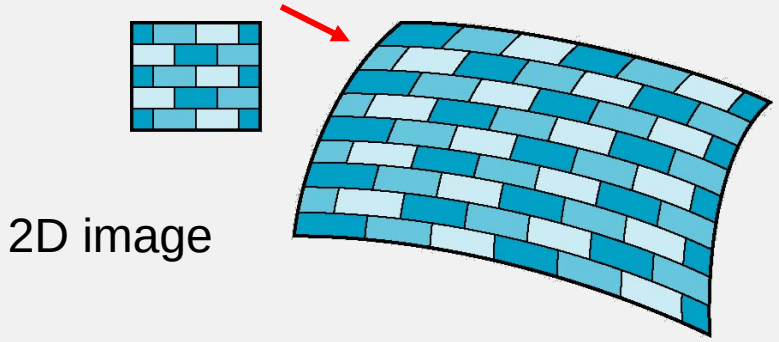
Modeling Object w/ Details

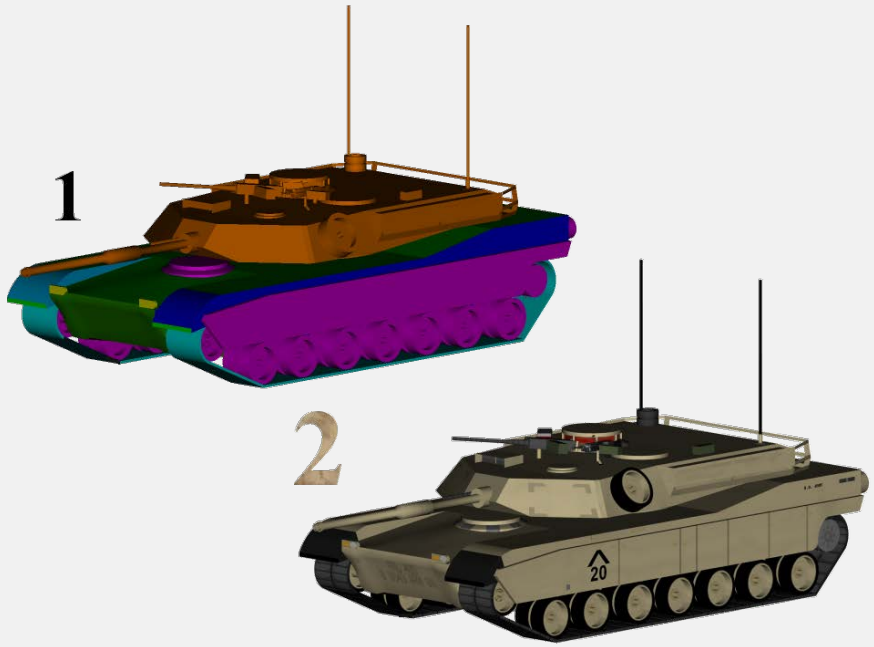
- How can you represent the details of an object?



Basic Idea of Texture Mapping

- Uses images to fill inside of polygons

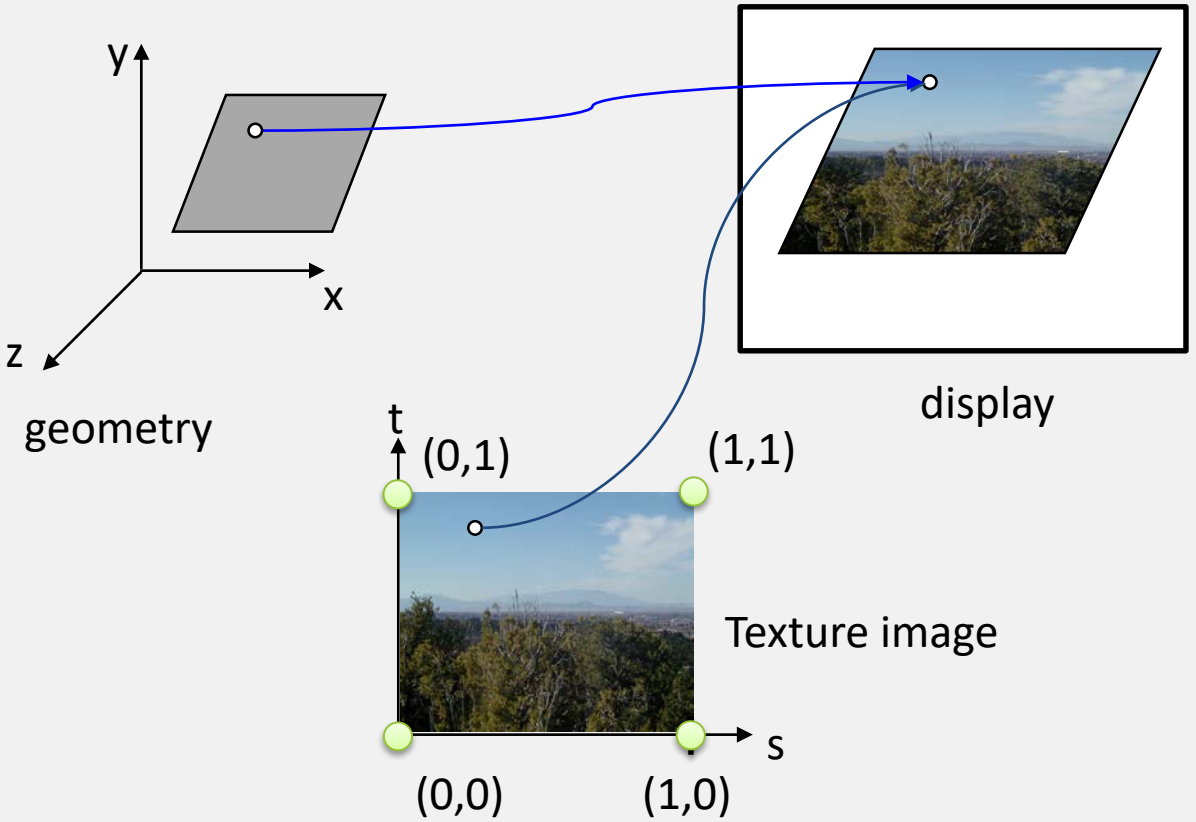




Texture

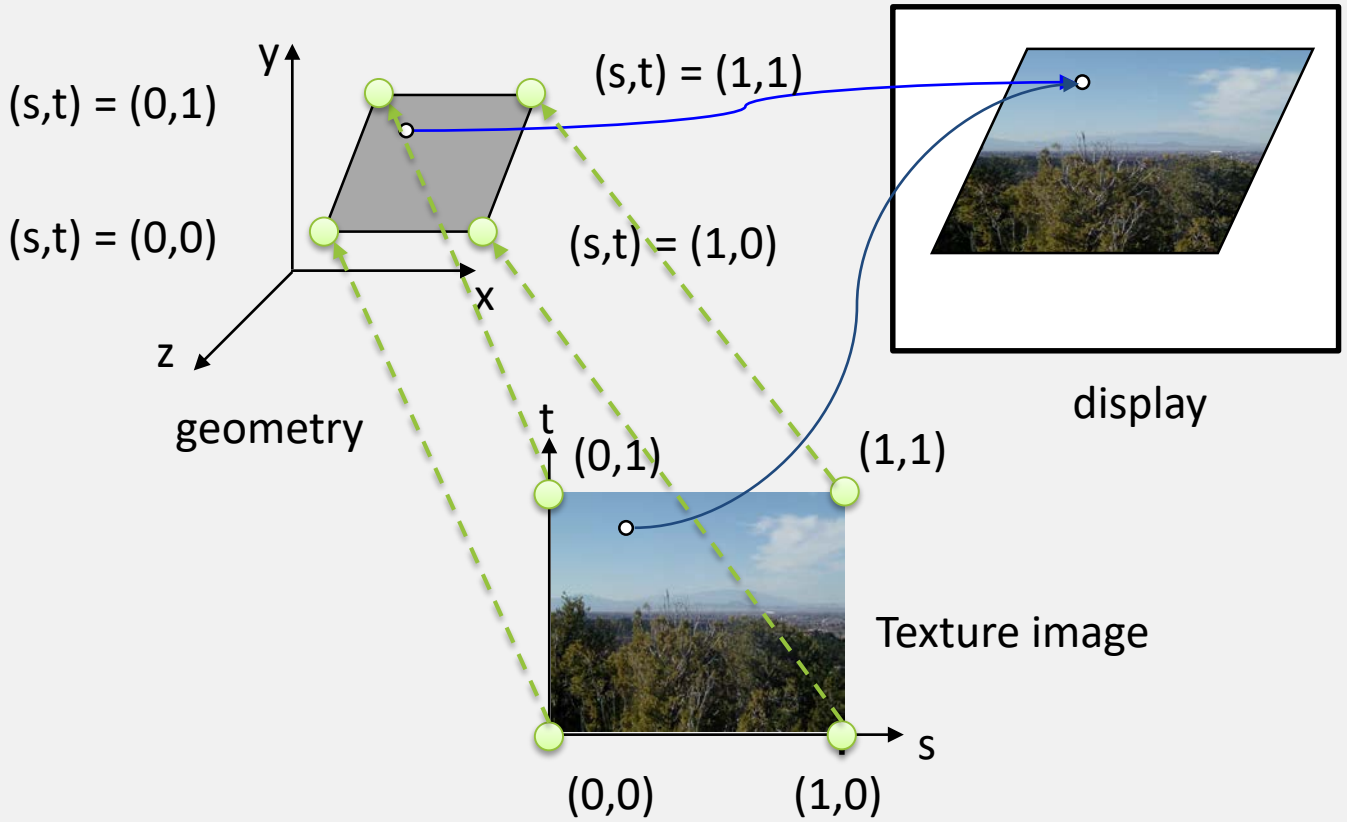
- Definition from Oxford dictionaries
  - The character of appearnce of a textile fabric as determined by the arrangement and thickness of its threads
- Definition used in computer graphics
  - Large chuncks of ***image data*** that can be used to paint the surfaces of objects

Texture Mapping

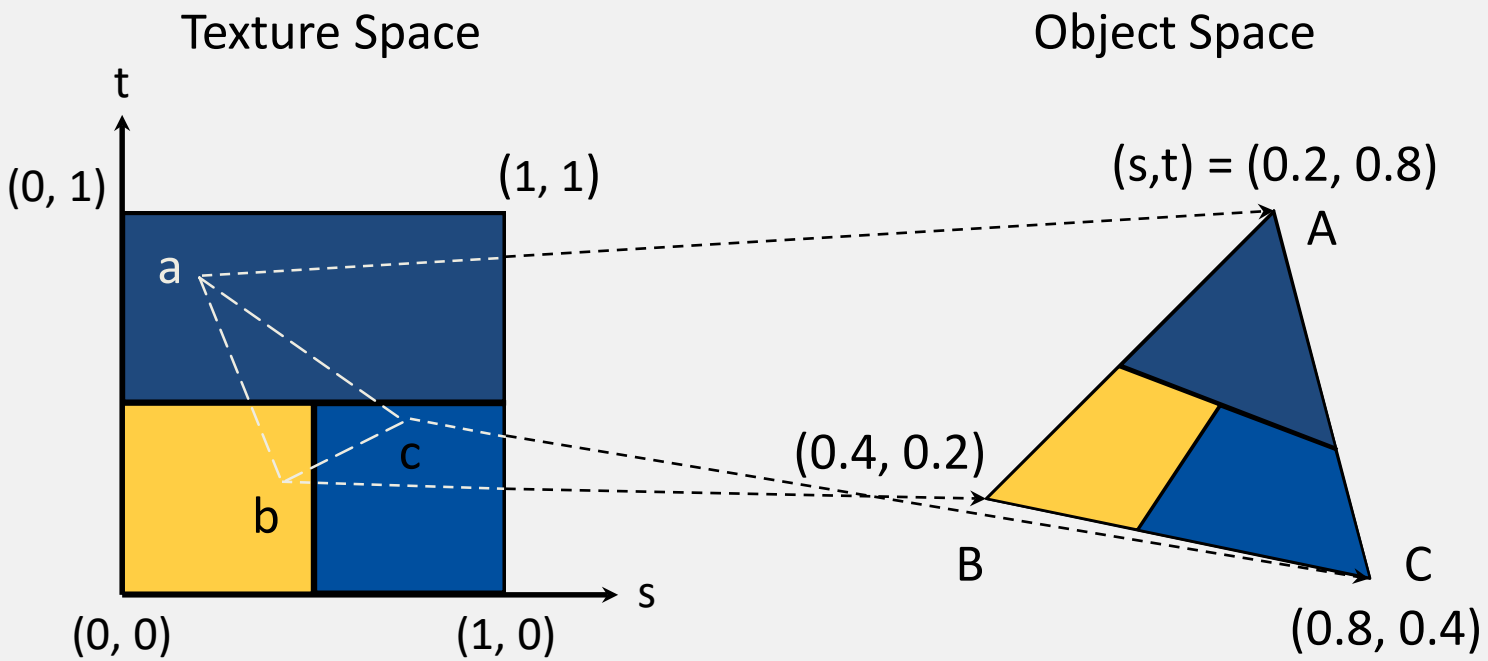


Texture Mapping

- We need to specify per-vertex texture coordinates

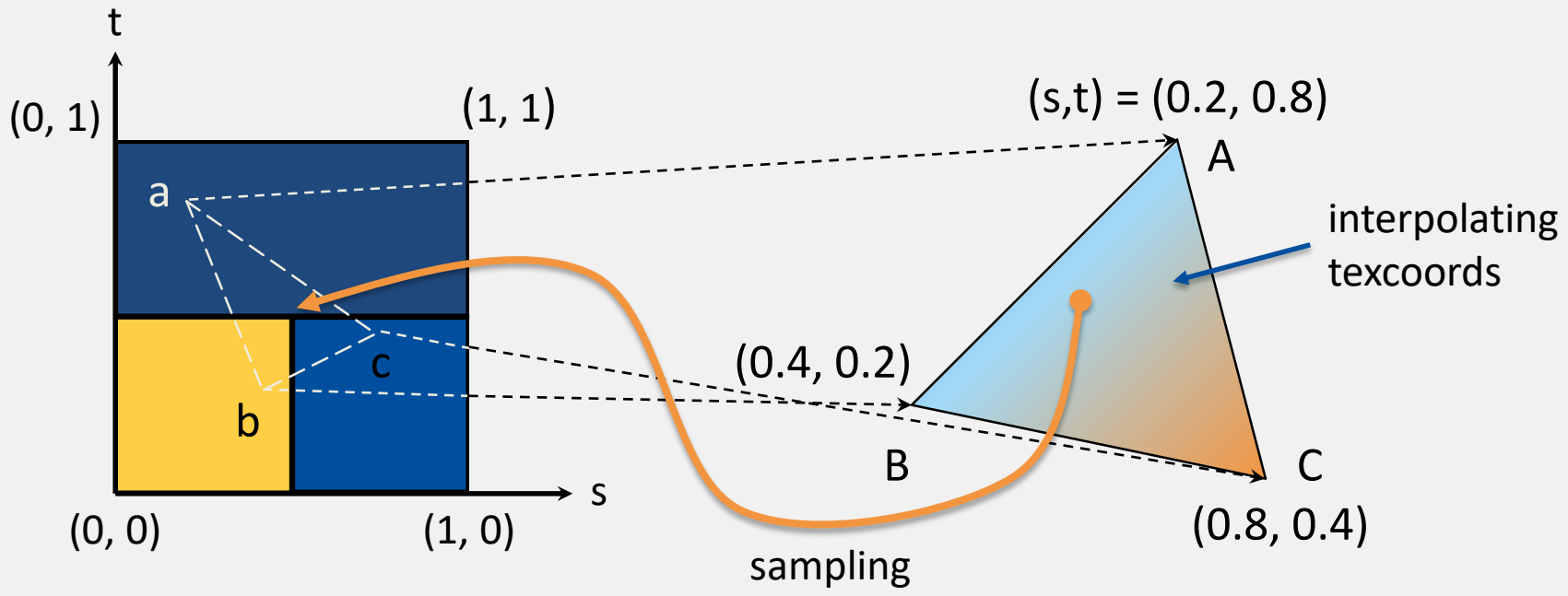


Texture Coordinates Matter



How can OpenGL ES Patch Colors from Texture?

- Bilinear interpolation on texture coordinates
- Sample colors from a texture image
  - Sampling problem!

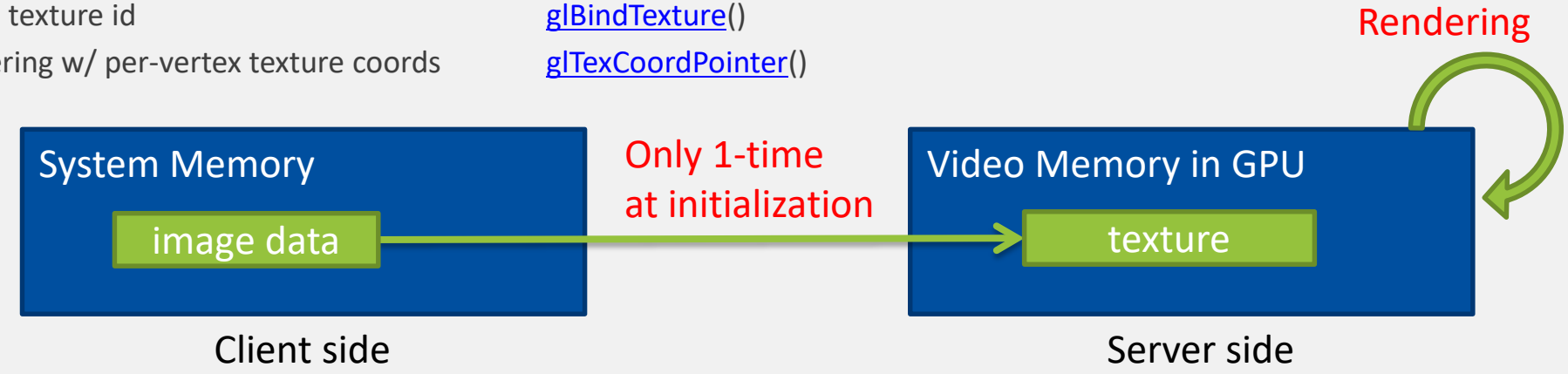


Demo – Texture Mapping

- Tutorial from Nate Robins
  - http://user.xmission.com/~nate/tutors.html

Using Texture Mapping in OpenGL ES

- Steps to use texture mapping
  1. Generate texture identifiers — glGenTextures()
  2. Binding a texture id — glBindTexture()
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters — glTexParameter()
       - Wrapping mode, Filtering methods
     - Specify texture data — glTexImage2D()
  4. Rendering with texture mapping
     - Enable texture mapping — glEnable(GL_TEXTURE_2D)
     - Bind a texture id — glBindTexture()
     - Rendering w/ per-vertex texture coords — glTexCoordPointer()



OpenGL ES codes – Texture Mapping

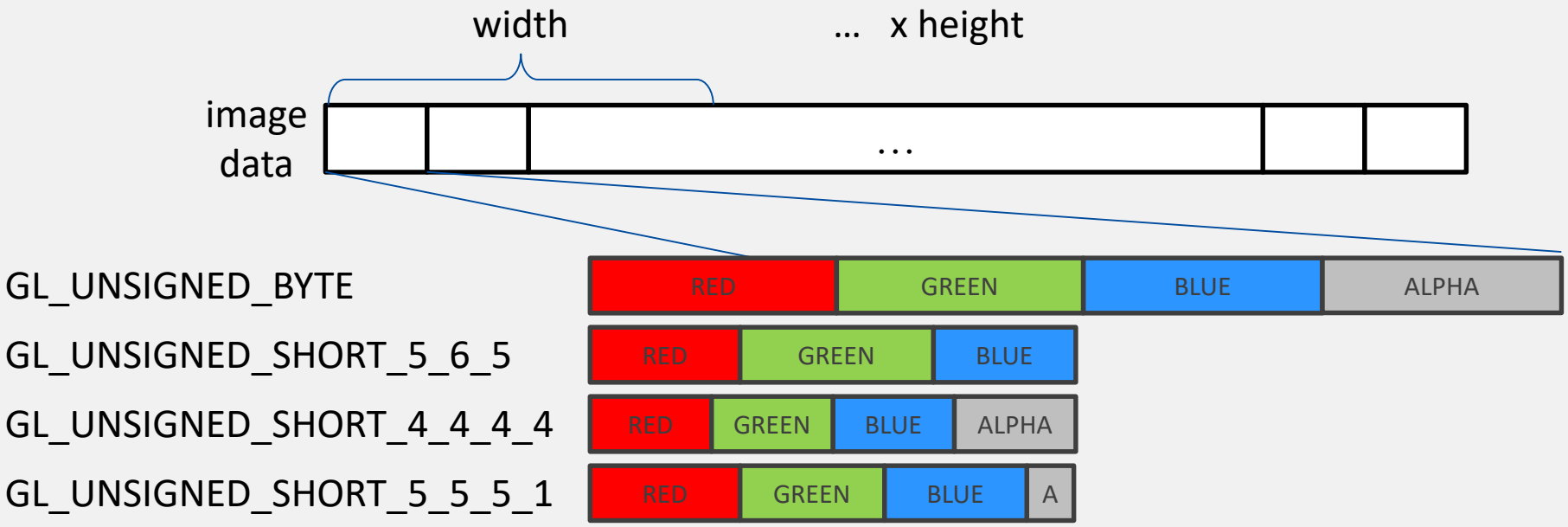
- Initialization
  1. Generate texture ids
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters
     - Wrapping mode, Filtering methods
     - Specify texture data
  4. Enable texture mapping
  5. Binding a texture id
  6. Rendering w/ texcoords

- OpenGL ES codes (C/C++)

```
// variables for texture mapping
GLuint        tex_id;
GLsizei       width, height;
GLbyte        *img_pixels;


// load an image from system
// img_pixels must locate the client-side memory of the image
// width/height should be update
// pixel format is important
// …

// Generate a texture
glGenBuffers(1, &tex_id);

// Bind a texture w/ the following OpenGL ES texture functions
glBindTexture(GL_TEXTURE_2D, tex_id);

// Set texture parameters (wrapping modes, sampling methods)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

// Transfer an image data in the client side to the server side
glTexImage2D(tex_id, 0, GL_RGBA, width, height, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, pixels);
```

Specifying a Texture Image

- Load an image from a file
  - There is no OpenGL function about it
    - You should use a platform-specific way
  - The image data should be stored in bitmap-like data structure
    - Data must be admitted by glTexImage2D()



OpenGL ES codes – Texture Mapping

- Rendering
  1. Generate texture ids
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters
     - Wrapping mode, Filtering methods
     - Specify texture data
  4. Enable texture mapping
  5. Binding a texture id
  6. Rendering w/ texcoords

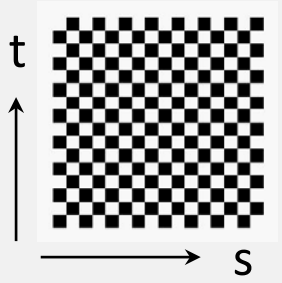
- OpenGL ES codes (C/C++)

```
// Enable texture mapping
glEnable(GL_TEXTURE_2D);

// Bind a texture w/ the following OpenGL ES texture functions
glBindTexture(GL_TEXTURE_2D, tex_id);


// Rendering w/ texcoords
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY);

glTexCoordPointer(…);
glVertexPointer(…);
glDrawArrays(…);

glDisableClientState(GL_TEXTURE_COORD_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

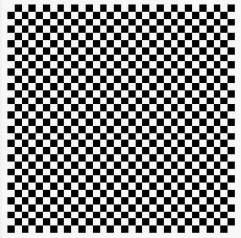
Texture Address Mode

Texture Address Mode

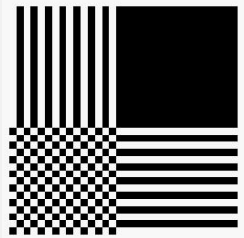
- How to repeat a given texture patterns when each texcoords *s* or *t* is not in [0,1]
- Why do we need it?



texture



GL_REPEAT
wrapping



GL_CLAMP_TO_EDGE
clamping

Texture Address Mode in OpenGL ES

- Steps to use texture mapping
  1. Generate texture identifiers
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - **Specify texture parameters**
       - **Wrapping mode**, Filtering methods
     - Specify texture data
  4. Rendering with texture mapping
     - Enable texture mapping
     - Bind a texture id
     - Rendering w/ per-vertex texture coords

Texture Address Mode in OpenGL ES

- Steps to use t
  1. Generate te
  2. Binding a te
  3. Specify text
     - Load ima
     - **Specify texture parameters**
       - **Wrapping mode**, Filtering methods
     - Specify texture data
  4. Rendering with texture mapping
     - Enable texture mapping
     - Bind a texture id
     - Rendering w/ per-vertex texture coords

```
// texture address mode as repeat
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);


// texture address mode as clamp
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
```

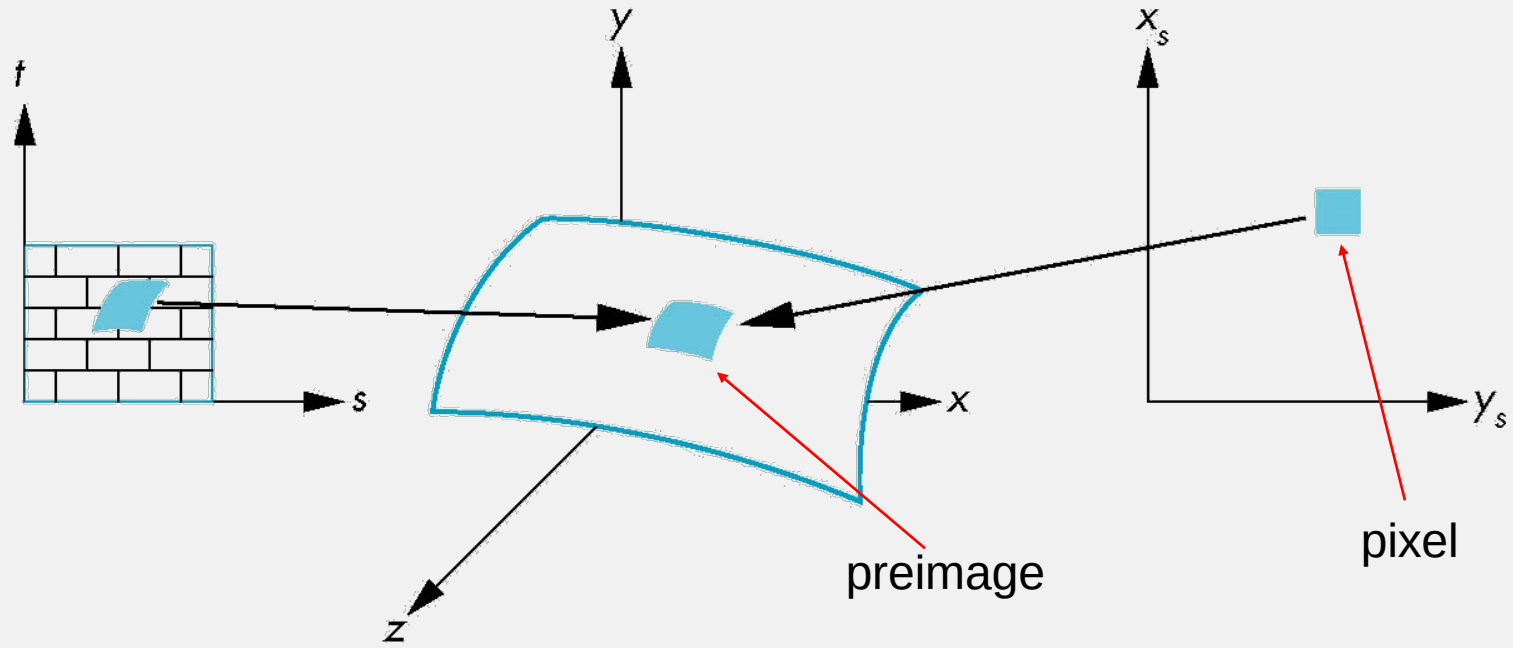
Demo – Texture Mapping

- Tutorial from Nate Robins
  - http://user.xmission.com/~nate/tutors.html

Sampling Problem in Texture Mapping

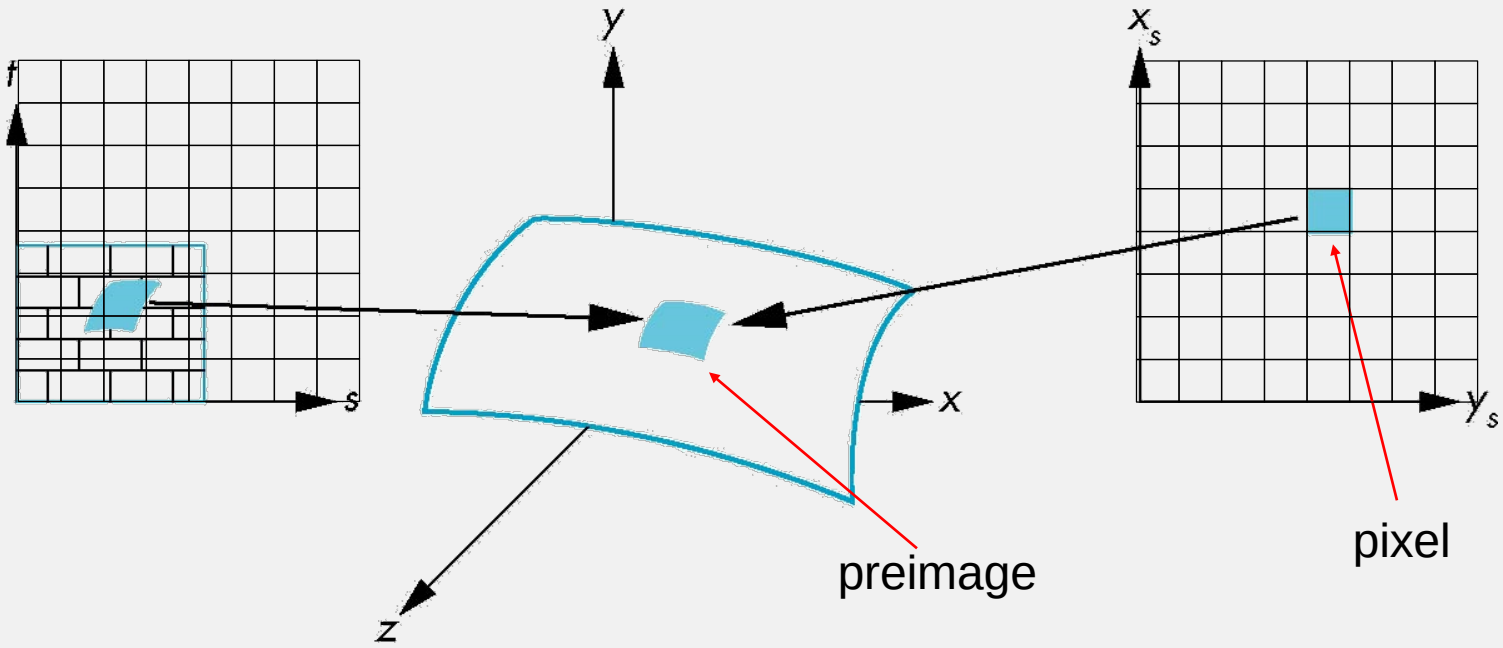
Sampling Problem

- A pixel must have one color value!!!

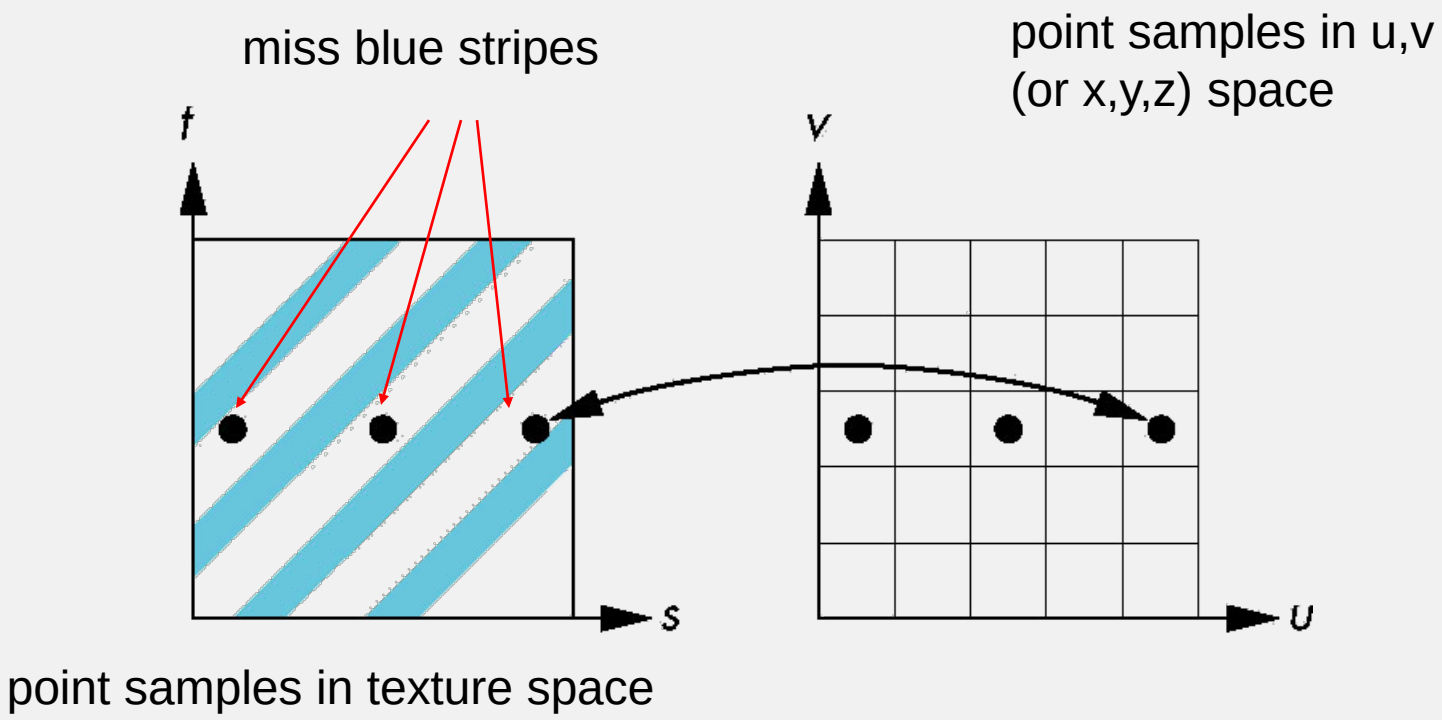


Sampling Problem

- A pixel must have one color value!!!
  - A pixel may correspond to several texels
  - A pixel may correspond to a small portion of a texel

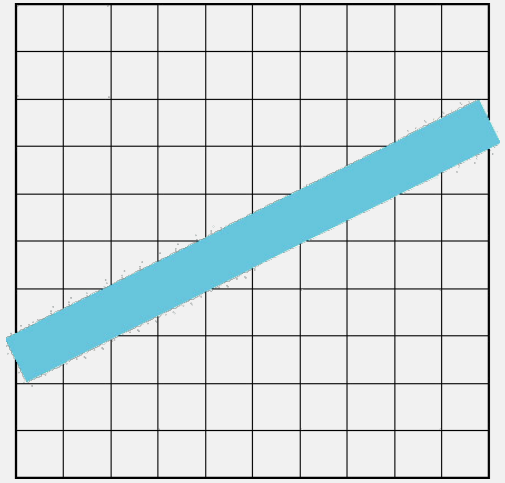


Aliasing

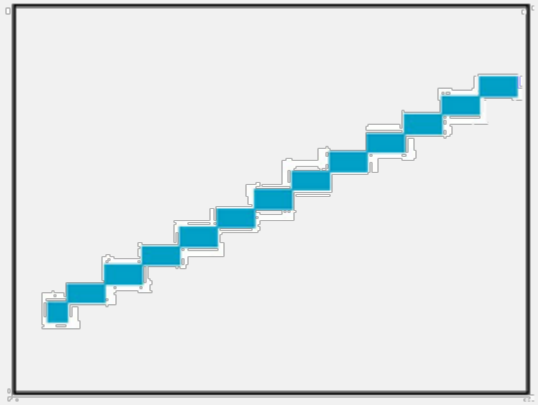

miss blue stripes
point samples in u,v
(or x,y,z) space
t
s
v
u
point samples in texture space

Aliasing

- What is aliasing?
  - Artifact from the limited sampling rates
  - What are antialiasing examples in the real-world? (video)



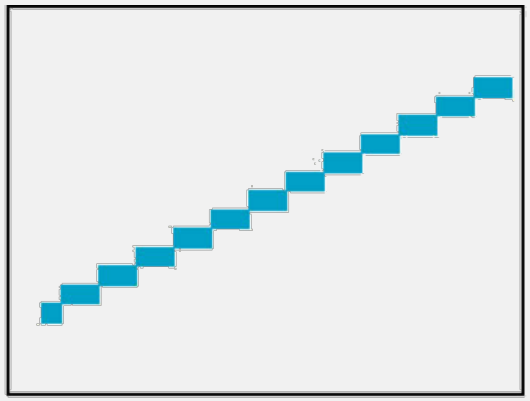
Ideal line



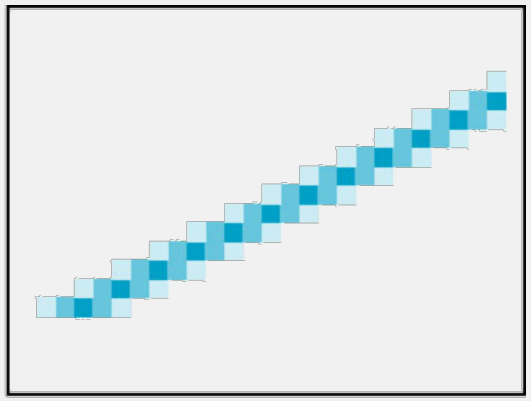
Rasterized line

Antialiasing

- Then, what is **anti**aliasing?
  - Ideally, it implies to cut-off the high-frequency terms.
  - In practice, it implies blurring



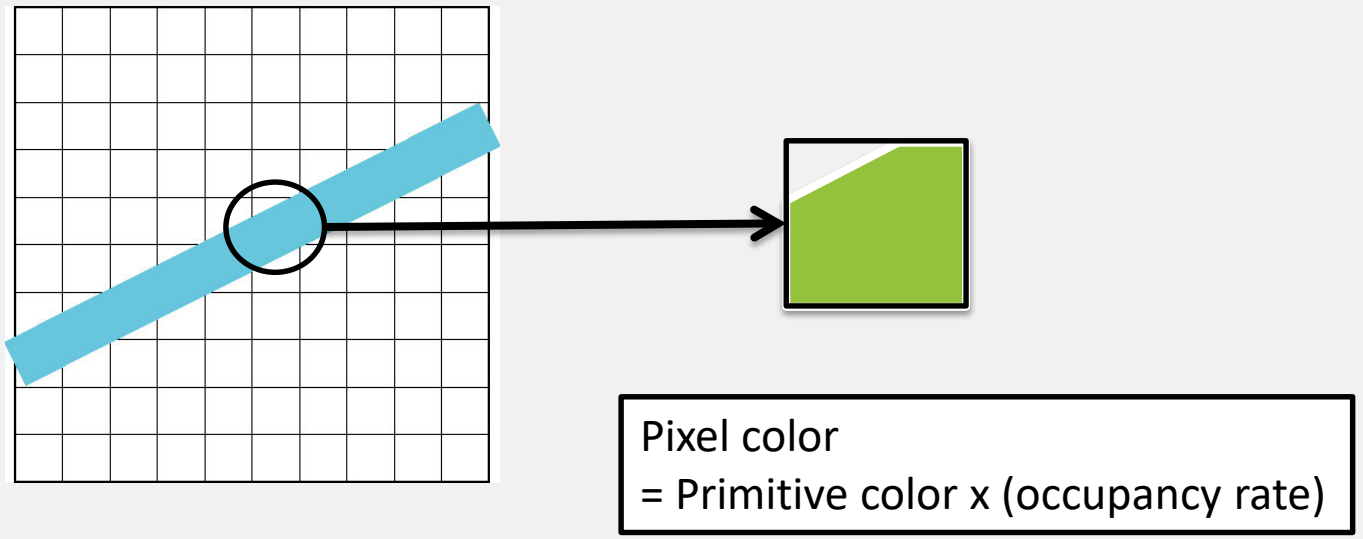
original



antialising

Antialising

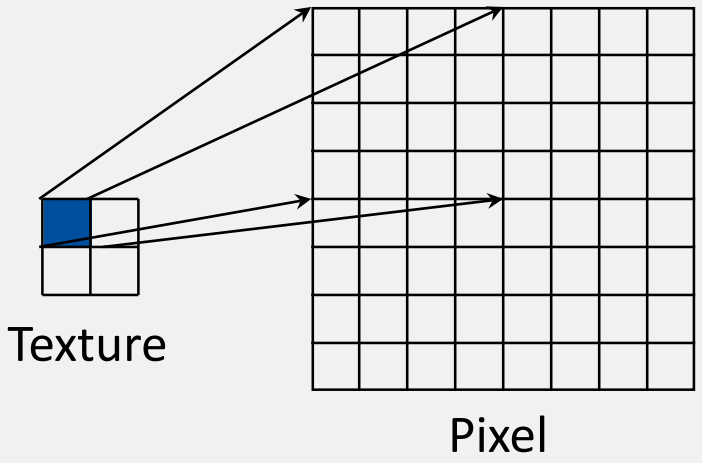
- Basic idea?
  - Consider the contribution of the primitive about each pixel



Magnification/Minification

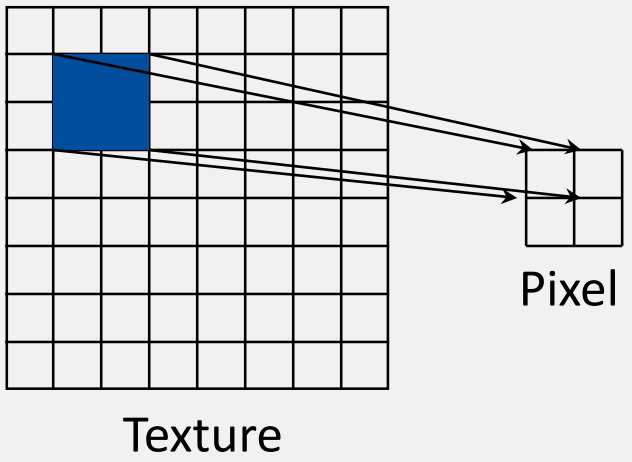
Magnification

- A texel is larger than 1 pixel
  - In general, zoom-in case



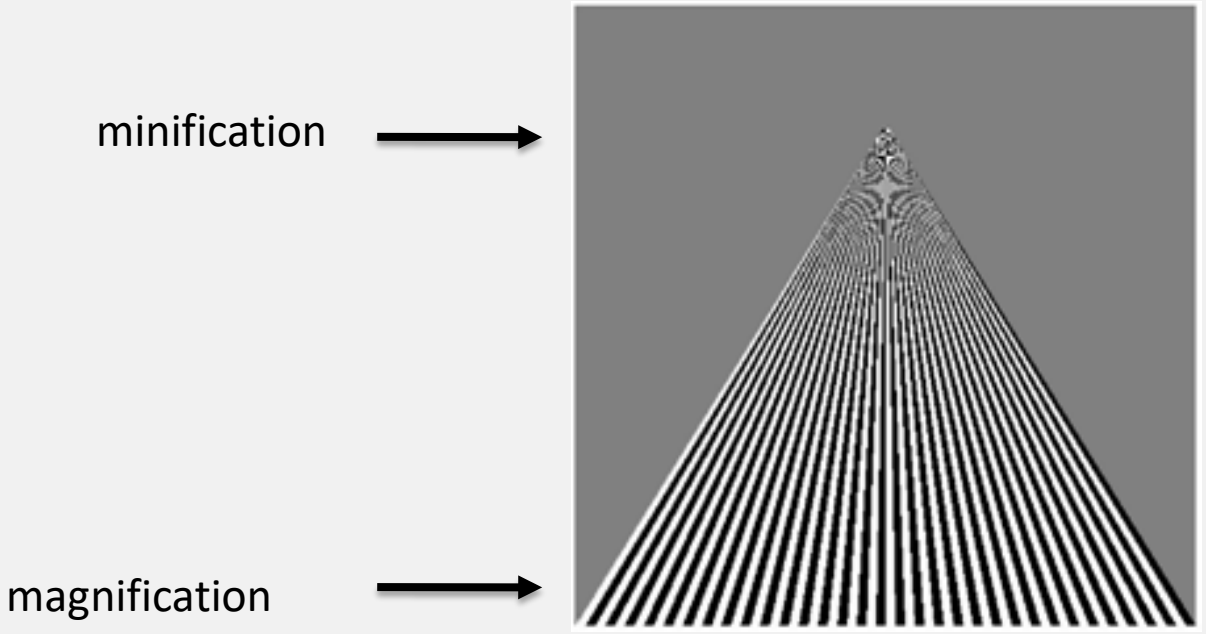
Minification

- A texel is smaller than 1 pixel
  - In general, zoom-out case



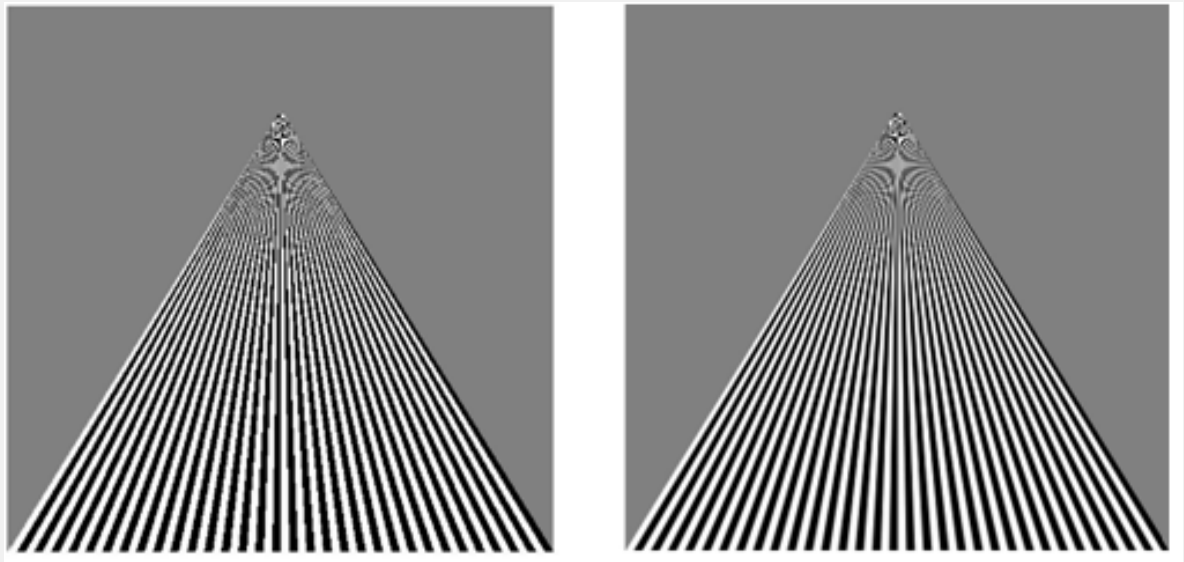
Example

- Aliasing happens both of the minification part and the magnification part



Texture Filtering

- Sampling patterns in textures
  - Nearest sampling (default)
  - Linear sampling (better quality)



Nearest sampling Linear sampling

Texture Filtering in OpenGL ES

- Steps to use texture mapping
  1. Generate texture identifiers
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - **Specify texture parameters** glTexParameter()
       - Wrapping mode, **Filtering methods**
     - Specify texture data
  4. Rendering with texture mapping
     - Enable texture mapping
     - Bind a texture id
     - Rendering w/ per-vertex texture coords

Texture Filtering in OpenGL ES

- Steps to use t
  1. Generate te
  2. Binding a te
  3. Specify text
     - Load ima
     - **Specify texture parameters**
       - Wrapping mode, **Filtering methods**
     - Specify texture data
  4. Rendering with texture mapping
     - Enable texture mapping
     - Bind a texture id
     - Rendering w/ per-vertex texture coords

```
// setting for nearest samplings
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);


// setting for linear samplings
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

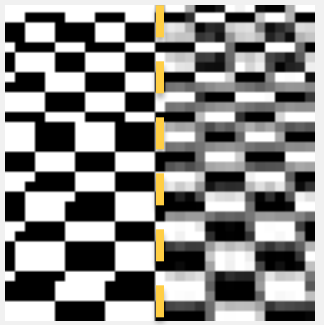
glTexParameter()

Advanced Topics of Texture Mapping

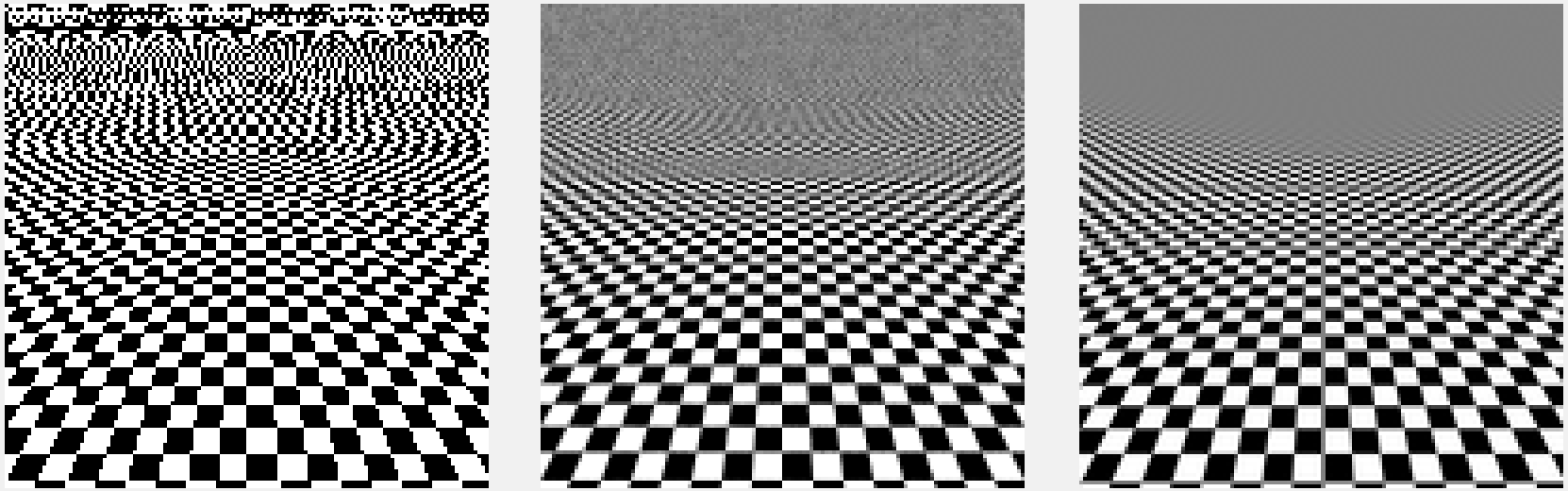
Advanced texture Sampling

- Aliasing of textures
  - Antialiasing is a kind of using blur filters

Nearest-point Filtering



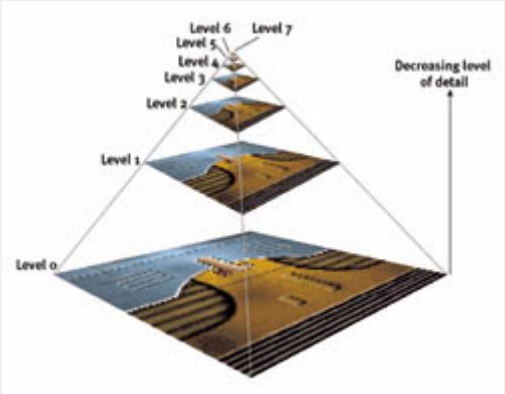
Linear Filtering



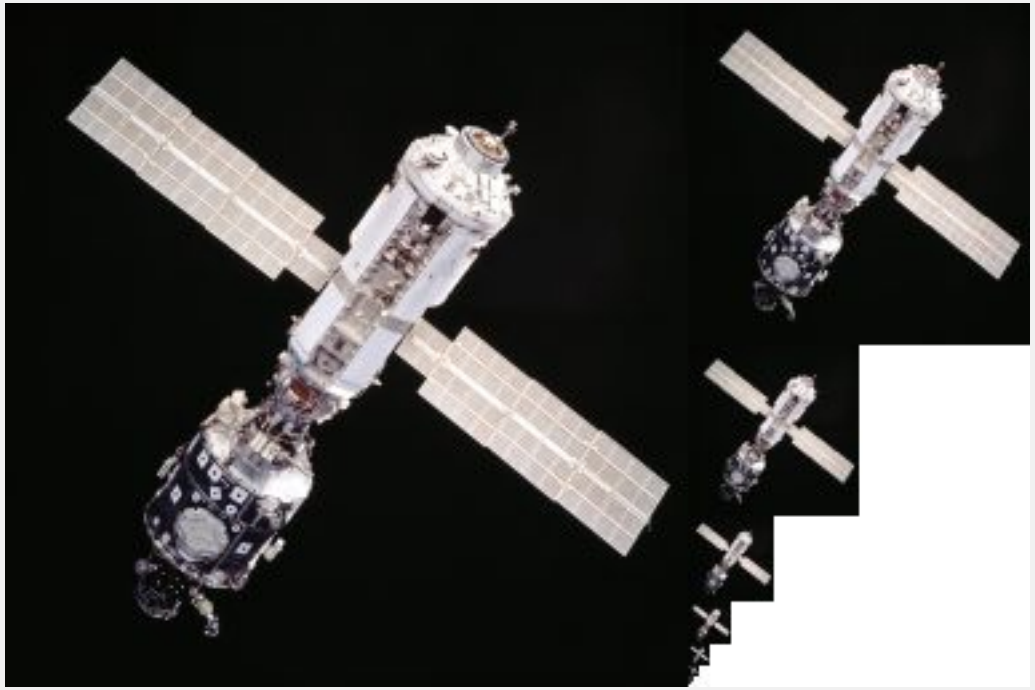
Advanced Filtering

Mipmap

- Efficient handling minification problem
- Building an image pyramid



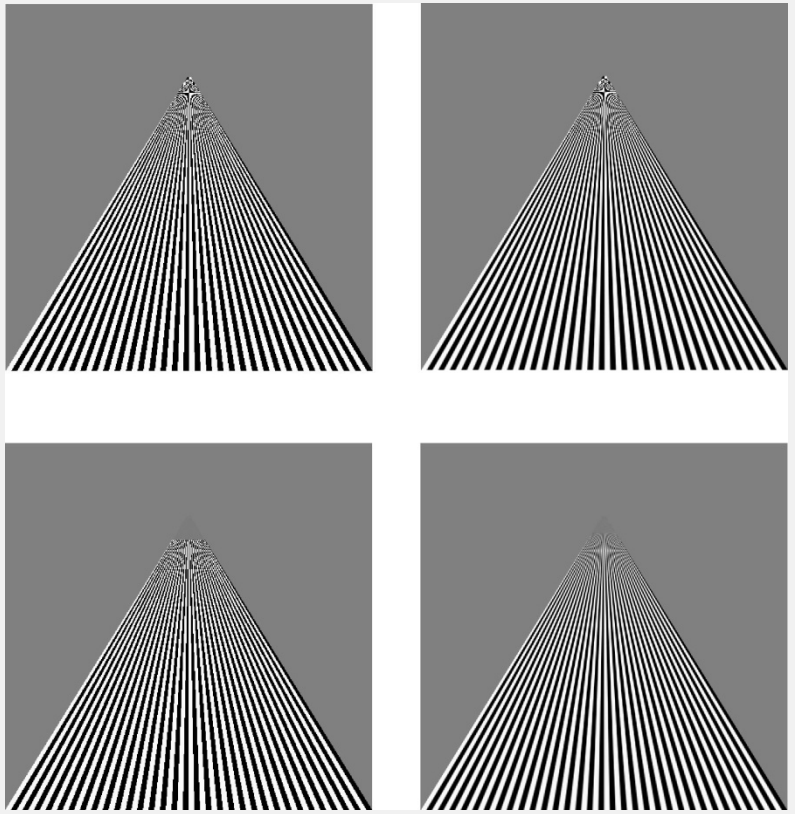
Concept of
image pyramid



Mipmap

- With mipmap, OpenGL use pre-filtered image (i.e., mipmap) for texture sampling

Point sampling



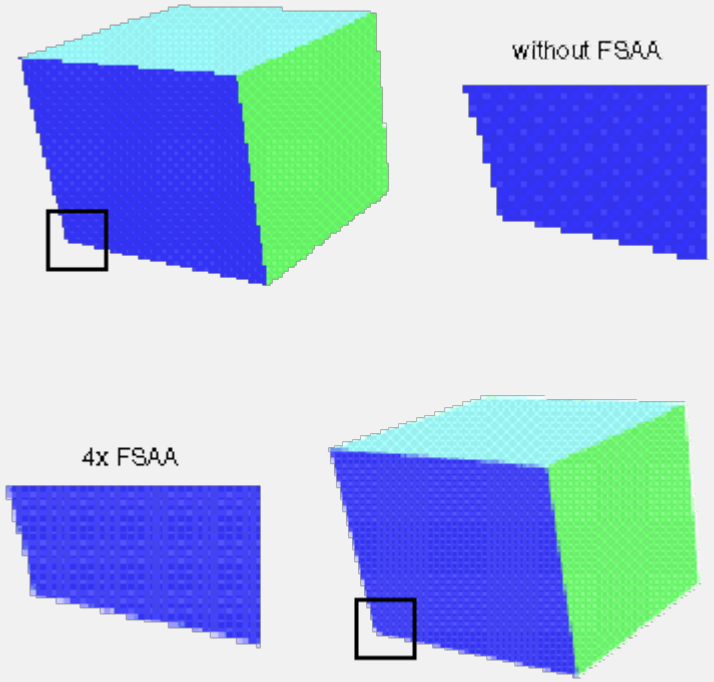
Linear filtering

Mipmapped
Point sampling

Mipmapped
Linear filtering

Full-Scene Anti-Aliasing (FSAA) or Multisample Anti-Aliasing (MSAA)

- Super sampling anti-aliasing for avoiding aliasing (or jaggies) on full-screen images

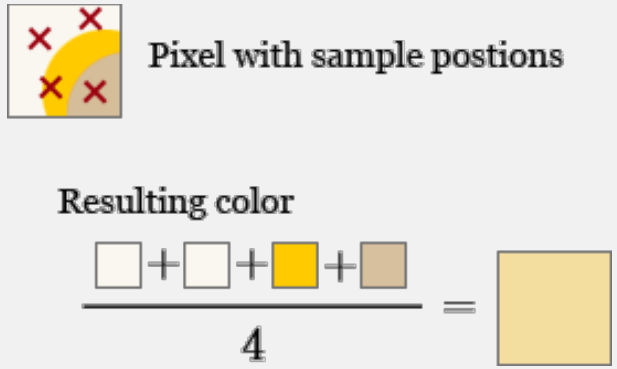


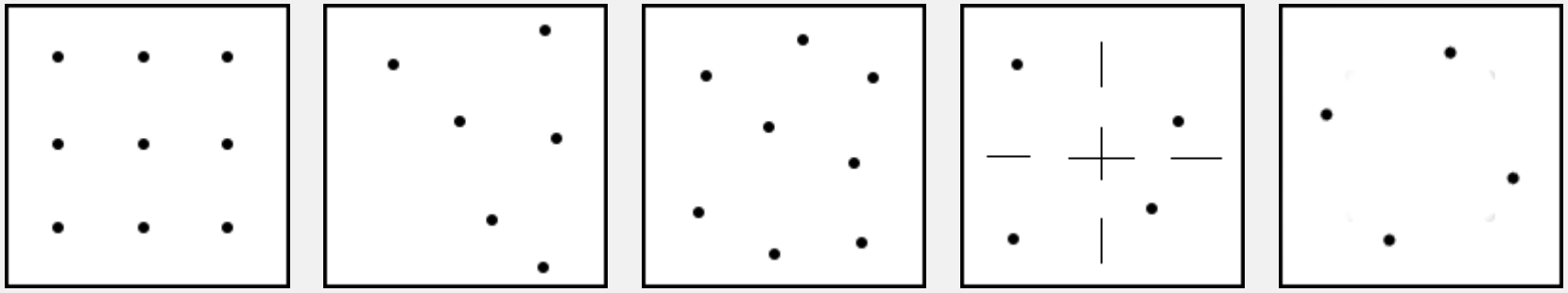
http://techpubs.sgi.com/library/dynaweb_docs/0650/SGI_EndUser/books/MPK_UG/sgi_html/ch04.html]

http://www.dansdata.com/prophet4500.htm

Full-Scene Anti-Aliasing (FSAA) or Multisample Anti-Aliasing (MSAA)

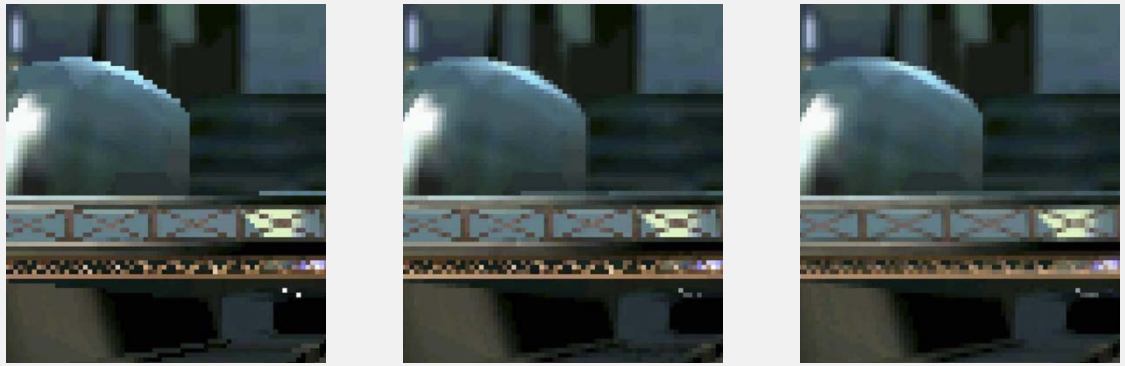
- Super sampling anti-aliasing for avoiding aliasing (or jaggies) on full-screen images





Full-Scene Anti-Aliasing (FSAA) or Multisample Anti-Aliasing (MSAA)

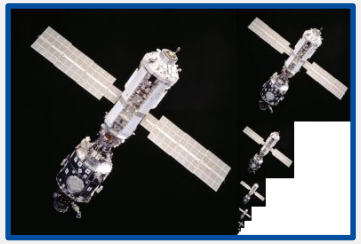
- FSAA invokes the issues about performance and power efficiency
  - GPU bendors (e.g., ARM Mali-T6XX series) argue that
    - 4x FSAA
      - Performance is matined at 90%+
      - Power-efficient anti-aliasing affectively comes ‘for free’ in the hardware
    - 16x FSAA
      - There is an up to 4x increase in per-pixel rendering cost
      - This is a cost-effective path to very high image quality



No FSAA　　4x FSAA　　16x FSAA

[images from ARM GPUs slides]

Using Mipmap in OpenGL ES



OpenGL ES 1.1

- Mipmap generation is NOT supported in the API level
  - Android: GLUtils package supports it

```
// Bind texture
gl.glBindTexture(GL10.GL_TEXTURE_2D, tex_id);

// Setting several texture parameters
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                   GL10.GL_TEXTURE_MAG_FILTER,
                   GL10.GL_LINEAR);
gl.glTexParameterf(GL10.GL_TEXTURE_2D,
                   GL10.GL_TEXTURE_MIN_FILTER,
                   GL10.GL_LINEAR_MIPMAP_NEAREST);
gl.glTexParameterf(GL11.GL_TEXTURE_2D,
                   GL11.GL_GENERATE_MIPMAP,
                   GL11.GL_TRUE);

// Generate a mipmap texture and
GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
```

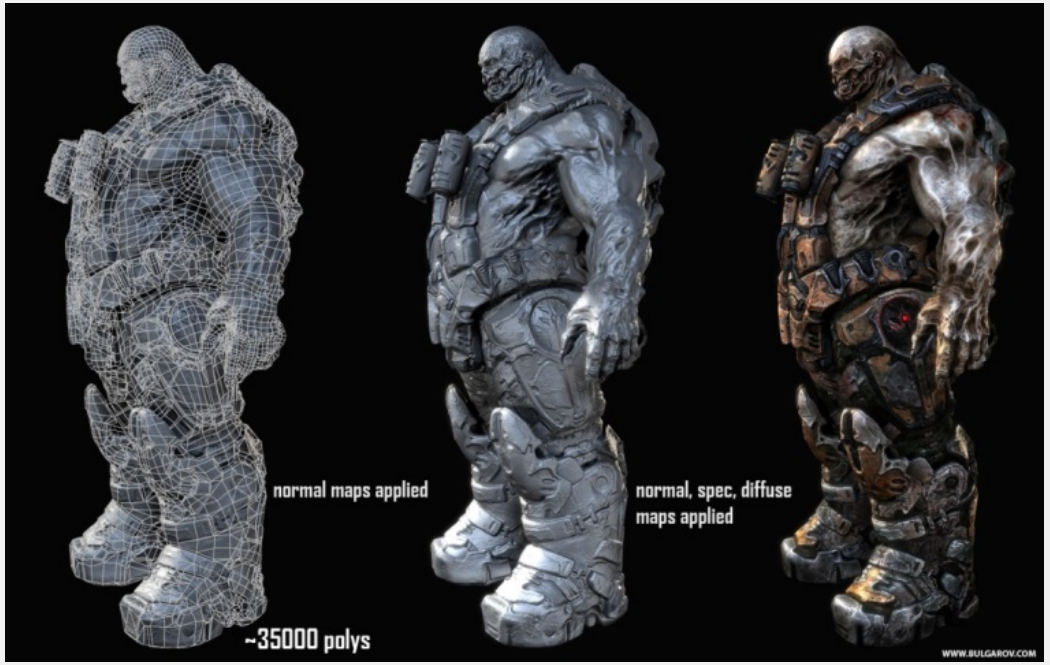
OpenGL ES 2.0 or higher

- Mipmap generation is supported in the API level
  - glGenerateMipmap()

```
// OpenGL function for generating a complete set of
// mipmaps for a texture object
//
// The target specifies the texture target of the
// active texture unit
//   MUST be GL_TEXTURE_2D or GL_TEXTURE_CUBE_MAP

void glGenerateMipmap (GLenum target);
```

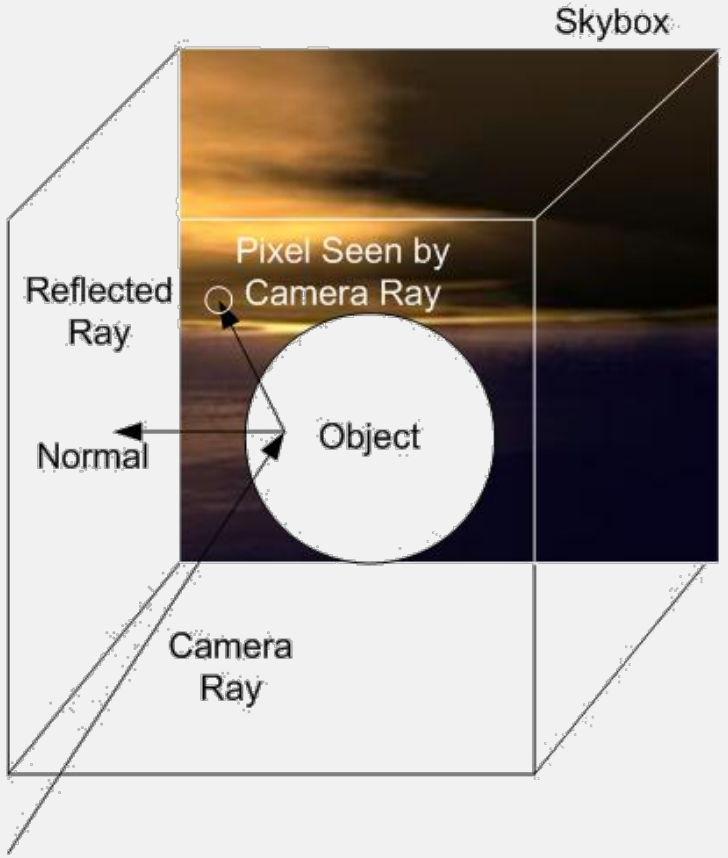
Mipmap – Texture Generation is important

- When using mipmap, we may encouter artifacts
  - Mipmap generation may blend texture colors in the other parts
  - Mipmap sampling may sample the ill-blended texture colors



Environment Mapping

[Forza Motorsport 4]

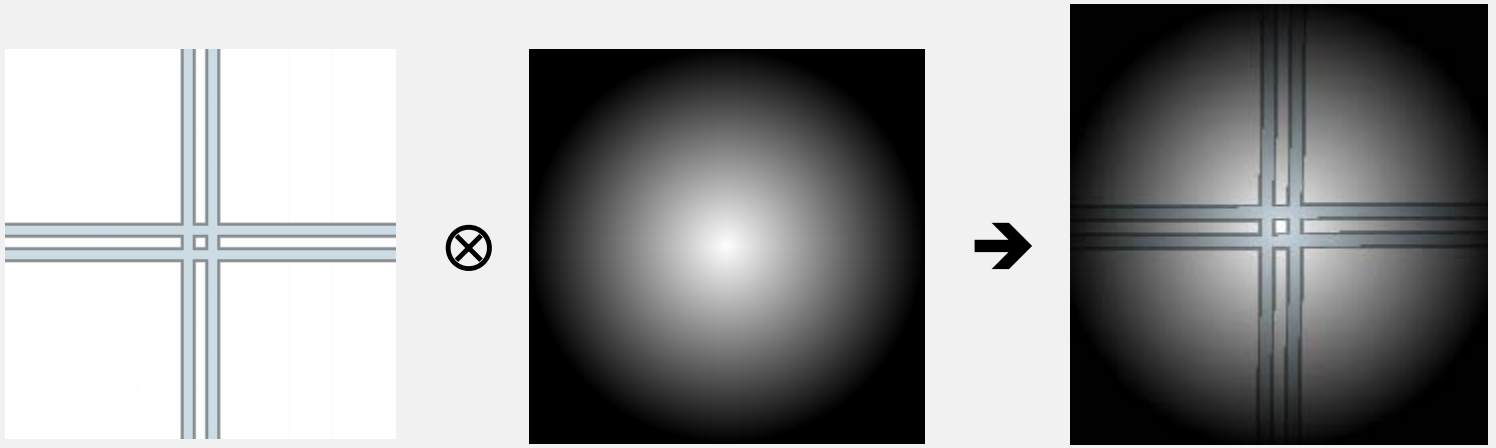


Normal Mapping

- Available w/ Pixel Shaders

Multi-Texturing

- Texture blending
  - To generate a new texture image by blending given two or more textures



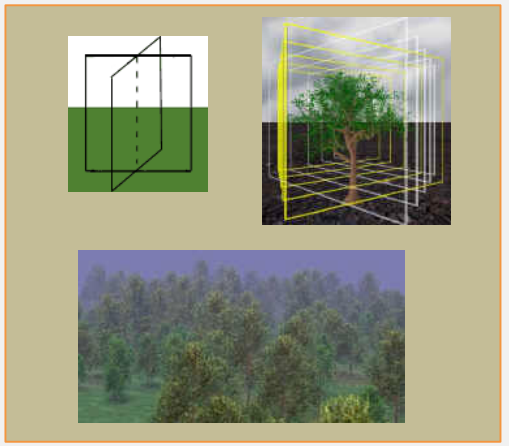
Multi-Texturing

- Quake 2
  - http://www.bigpanda.com/trinity/article1.html

Blending

Alpha Blending

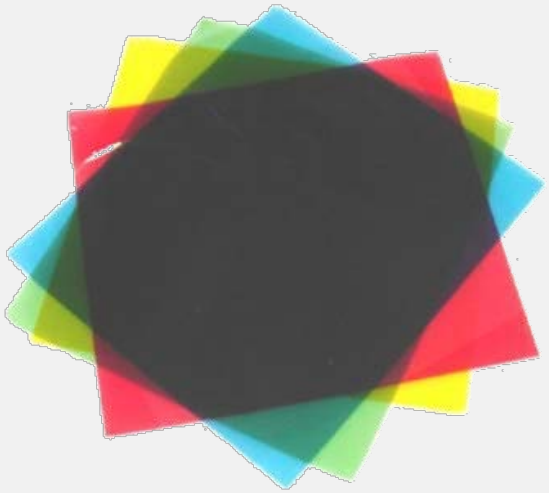
- Billboard
  - To represent a 3D object with a textured 2D plane
  - Texture images must have alpha channels
- Alpha value represents opacity
  - 1: opque
  - 0: transparent

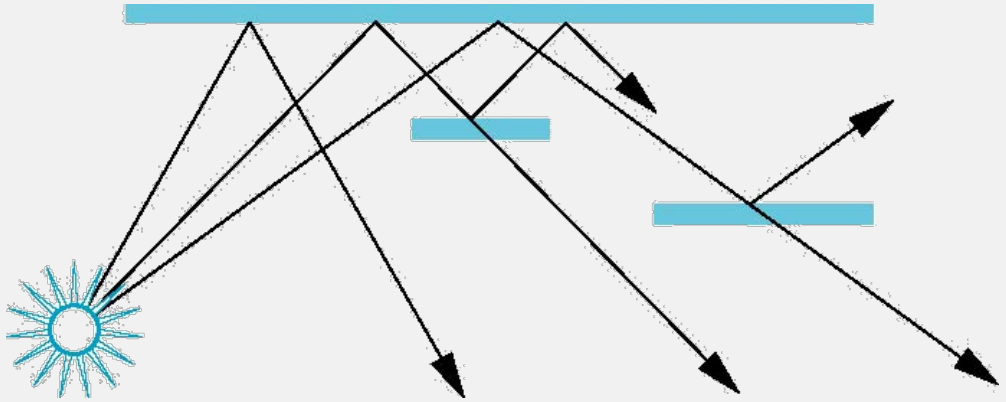


[from google image]

Alpha Blending

- Physically correct model
  - We need to consider complicated inter-surface reflections
  - Difficult in real-time

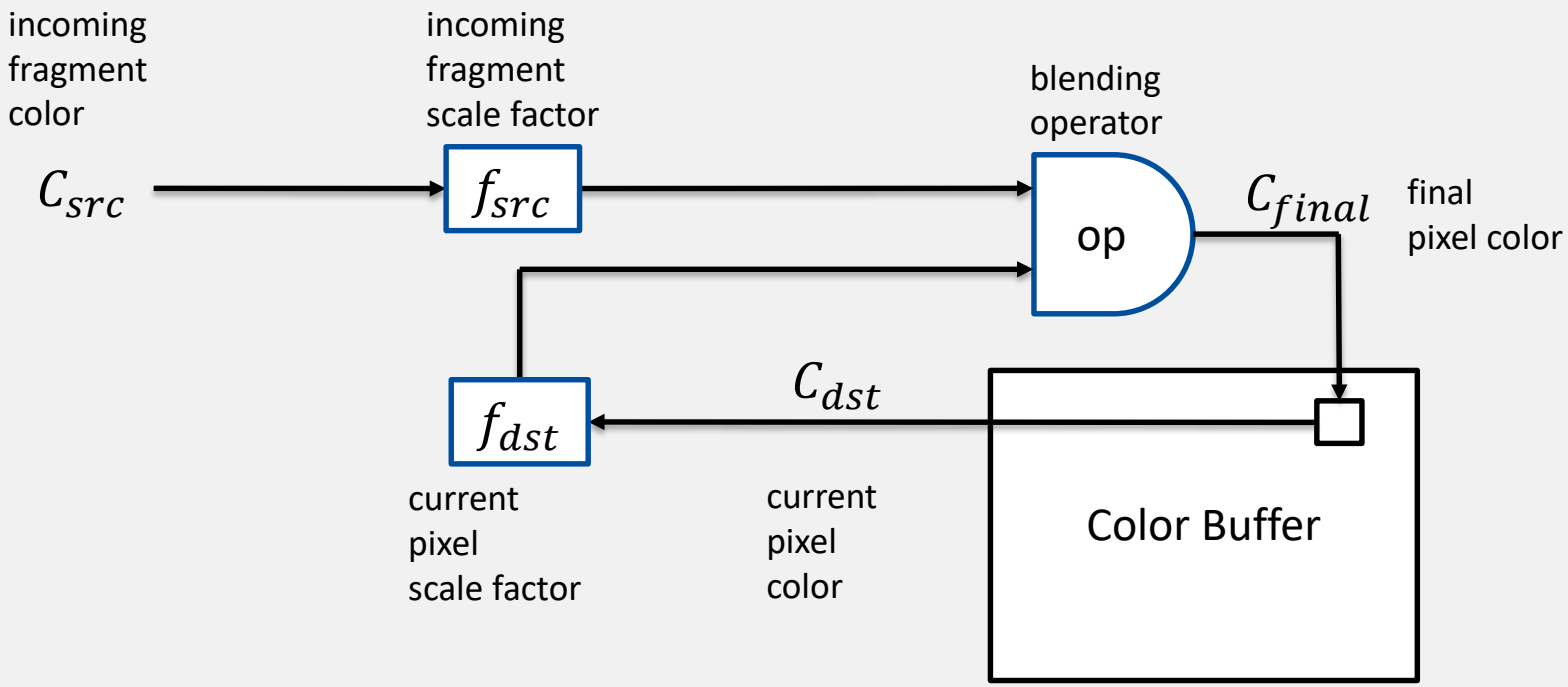




Alpha Blending

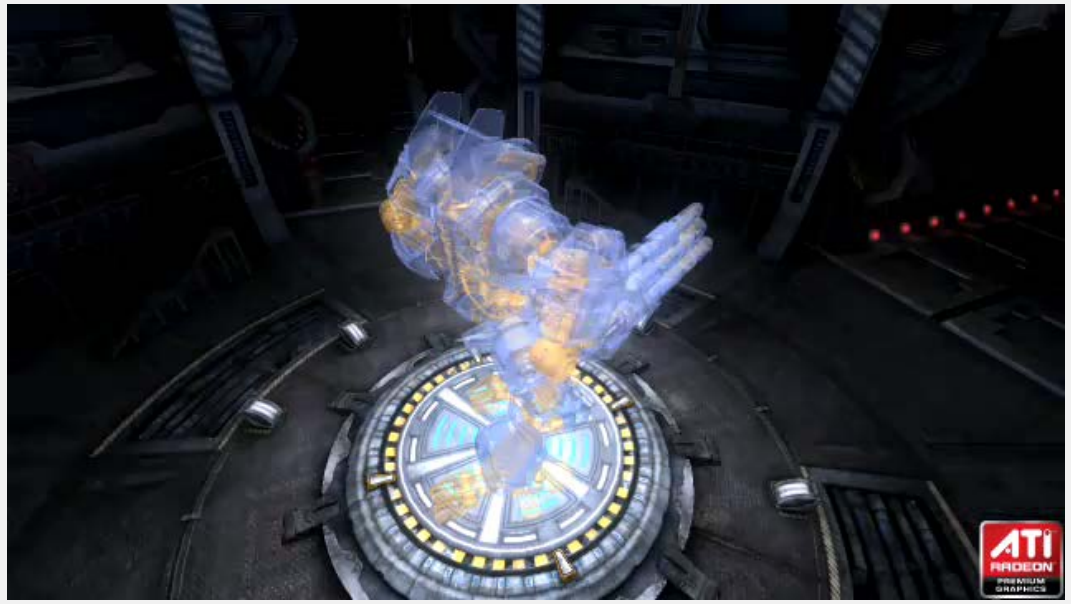
- Blending equation in OpenGL ES

$$C_{final} = f_{src}C_{src} \text{ op } f_{dst}C_{dst}$$



Alpha Blending – Painter's Algorithm

- We render polygons a back to front order for alpha blending
  - Even though graphics HW supports the z-buffer algorithm, we should use painter's algorithm for alpha blending

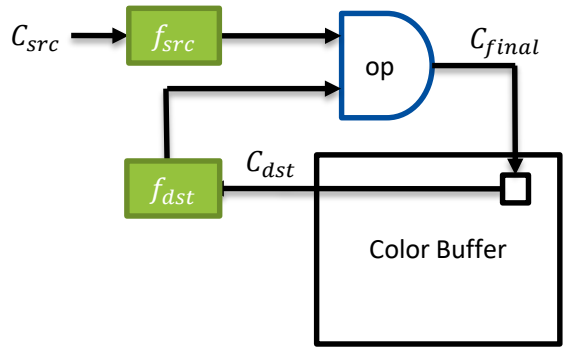


[AMD DirectX 11 Demo for H/W accelerated alpha blending]
(video, youtube)

Alpha Blending

- glBlendFunc(): specify pixel arithmetic
- glBlendFuncSeperate(): specify pixel arithmetic for RGB / A separately

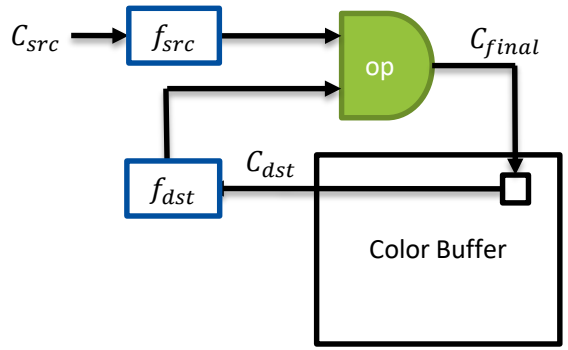
```
// sfactor  specifies how the red, green, blue, and alpha source blending factors are computed.
// dfactor  specifies how the red, green, blue, and alpha destination blending factors are computed.
//
//          Parameters                      RGBA blending factors
//          GL_ZERO                         0 0 0 0
//          GL_ONE                          1 1 1 1
//          GL_SRC_COLOR                    Rs Gs Bs As
//          GL_ONE_MINUS_SRC_COLOR          1-Rs 1-Gs 1-Bs 1-As
//          GL_SRC_ALPHA                    As As As As
//          GL_ONE_MINUS_SRC_ALPHA          1-As 1-As 1-As 1-As
//          GL_DST_COLOR                    Rd Gd Bd Ad
//          GL_ONE_MINUS_DST_COLOR          1-Rd 1-Gd 1-Bd 1-Ad
//          GL_DST_ALPHA                    Ad Ad Ad Ad
//          GL_ONE_MINUS_DST_ALPHA          1-Ad 1-Ad 1-Ad 1-Ad
//          GL_CONSTANT_COLOR               Rc Gc Bc Ac             constant blending color
//          GL_ONE_MINUS_CONSTANT_COLOR     1-Rc 1-Gc 1-Bc 1-Ac     (Rc, Gc, Bc, Ac)
//          GL_CONSTANT_ALPHA               Ac Ac Ac Ac             comes from
//          GL_ONE_MINUS_CONSTANT_ALPHA     1-Ac 1-Ac 1-Ac 1-Ac     glBlendColor()
//          GL_SRC_ALPHA_SATURATE           min(As, 1-Ad)  1

void glBlendFunc(GLenum sfactor, GLenum dfactor);
void glBlendFuncSeparate(GLenum srcRGB, GLenum dstRGB, GLenum srcAlpha, GLenum dstAlpha);
```



Alpha Blending

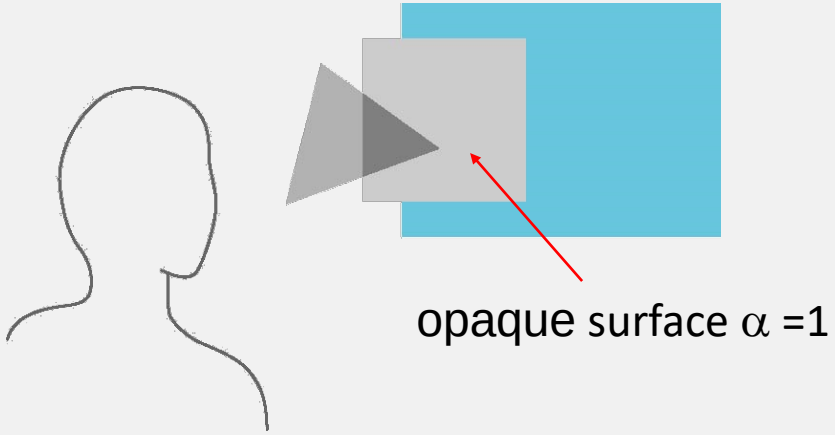
- glBlendEquation(): specify the blend equation for RGBA
- glBlendEquationSeperate(): specify the blend equations for RGB / A separately

```
// mode     specifies how source and destination colors are combined.
//
//          Parameters                          Equation
//          GL_FUNC_ADD                         SRC + DST
//          GL_FUNC_SUBTRACT                    SRC - DST
//          GL_FUNC_REVERSE_SUBTRACT            DST - SRC
//
// For these equations all color components are understood
// to have values in the range [0, 1].
// The results of these equations are clamped to the range [0, 1].



void glBlendEquation(GLenum mode);
void glBlendEquationSeparate(GLenum modeRGB, GLenum modeAlpha);
```



Alpha Blending

- Alpha Blending in OpenGL
  - z-buffer test?
  - Rendering order about several objects?
    - Blending functions are order dependent
    - Opaque polygons block all polygos behind them and affect the depth buffer
    - Translucent polygons should not affect depth buffer
    - Sort polygons first to remove order dependency



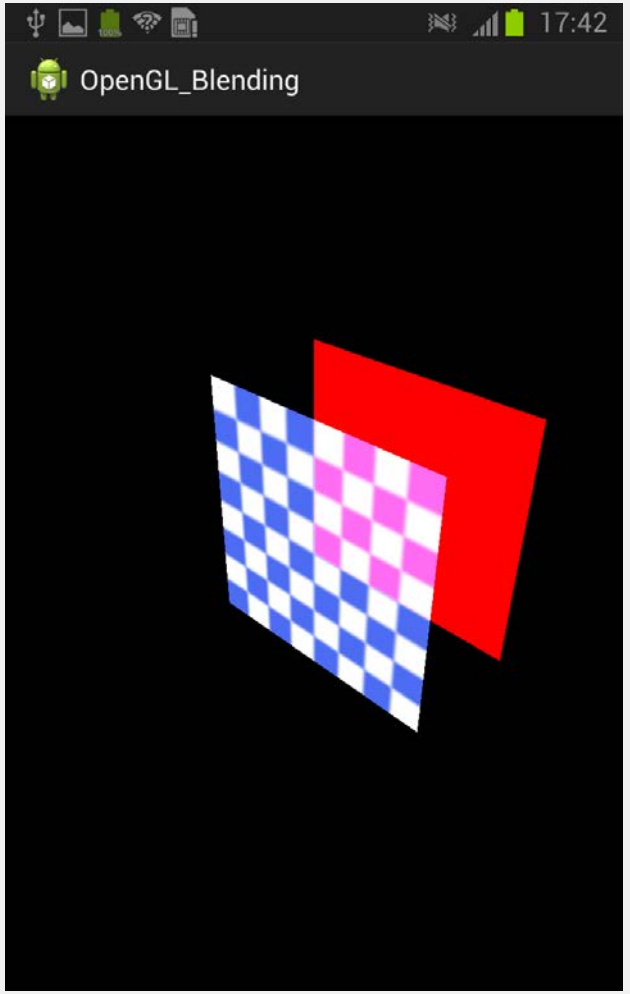
Using Alpha Blending in OpenGL ES

- Set Pixelformt of frame buffer as RGBA
- Draw an Object with alpha channel
  - When using texture, texel should have alpha channel
- Disable depth test in OpenGL
  - glDepthMask(GL_FALSE) or glDisable(GL_DEPTH_TEST);
- Enable blending function in OpenGL
  - glBlendFunc(…) or glBlendFuncSeparate(…)
  - glBlendEquation(…) or glBlendEquationSeparate(…)
  - glEnable(GL_BLEND);
- Draw objects with a carefully selected order
- Disable blending function in OpenGL
  - glDisable(GL_BLEND);
- Enable depth test in OpenGL
  - glDepthMask(GL_TRUE) or glEnable(GL_DEPTH_TEST);

Practice – Alpha Blending

- Multiple shader program
  - Simple shader
  - Texture shader

- Alpha blending
  - Using image alpha bits



Alpha blending

Simple Shader

- Uniform / attribute
  - model-view-projection matrix
  - vertex
- Vertex shader
  - Transform vertex
- Fragment shader
  - Set fragment color

Vertex / Fragment Shaders

```
uniform mat4 mvp_matrix;
attribute vec4 a_vertex;

void main() {
  gl_Position = mvp_matrix * a_vertex;
}
```

```
precision mediump float;

void main() {
  gl_FragColor = vec4(1,0,0,1);
}
```

Alpha blending

Texture mapping

- Uniform / attribute
  - model-view-projection matrix
  - Vertex, texture coordinate
- Vertex shader
  - Transform vertex
  - Send texture coordinate
- Fragment shader
  - Access a texture using sampler
  - Set fragment color

Vertex / Fragment Shaders

```
uniform mat4 mvp_matrix;

attribute vec4 a_vertex;
attribute vec2 a_texcoord;

varying vec2 v_texcoord;

void main() {
  v_texcoord = a_texcoord;
  gl_Position = mvp_matrix * a_vertex;
}
```

```
precision mediump float;

uniform sampler2D image;

varying vec2 v_texcoord;

void main() {
  gl_FragColor = texture2D(image, v_texcoord);
}
```

Alpha blending

Alpha blending setting

- Disable depth test
- Enable blending
- Specify blending function

- Create shader programs
  - Simple / texture shader

Android codes

```
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
  GLES20.glDisable(GLES20.GL_DEPTH_TEST);
  GLES20.glEnable(GLES20.GL_BLEND);
  GLES20.glBlendFunc(GLES20.GL_ONE, GLES20.GL_ONE);

  vertex_buffer = BufferUtil.makeFloatBuffer(vertices);
  texcoord_buffer = BufferUtil.makeFloatBuffer(texcoords);

  // create simple shaders
  simple_program = createProgram(simple_vertex_shader_src,
simple_fragment_shader_src);
  if (simple_program == 0) {
    throw new RuntimeException("Could not create simple shader program");
  }

  // ...

  // create texture shaders
  texture_program = createProgram(texture_vertex_shader_src,
texture_fragment_shader_src);
  if (texture_program == 0) {
    throw new RuntimeException("Could not create texture shader program");
  }

  // ...
```

Alpha blending

First path

- Render with simple shader program
- Translate a rectangle

Android codes

```
// draw
// first path
GLES20.glUseProgram(simple_program);

Matrix.setIdentityM(model_matrix, 0);
Matrix.translateM(model_matrix, 0, 0, 0,-1);
Matrix.multiplyMM(model_view_matrix, 0, view_matrix, 0,
model_matrix, 0);
Matrix.multiplyMM(model_view_projection_matrix, 0,
projection_matrix, 0, model_view_matrix, 0);

GLES20.glUniformMatrix4fv(simple_mvp_matrix_handle, 1,
false, model_view_projection_matrix, 0);

GLES20.glVertexAttribPointer(simple_vertex_handle, 3,
GLES20.GL_FLOAT, false, 0, vertex_buffer);
GLES20.glEnableVertexAttribArray(simple_vertex_handle);

GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);

GLES20.glUseProgram(0);
```

Alpha blending

Second path

- Render with texture shader program
- Draw a rectangle with texture

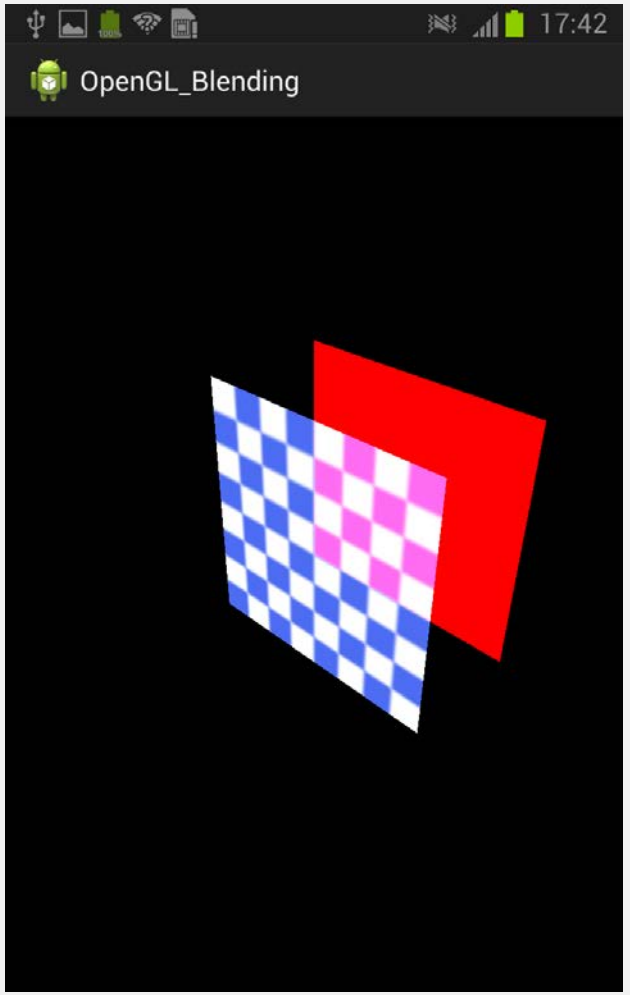
Android codes

```
// second path
GLES20.glUseProgram(texture_program);

Matrix.setIdentityM(model_matrix, 0);
Matrix.multiplyMM(model_view_matrix, 0, view_matrix, 0,
model_matrix, 0);
Matrix.multiplyMM(model_view_projection_matrix, 0,
projection_matrix, 0, model_view_matrix, 0);

GLES20.glUniformMatrix4fv(texture_mvp_matrix_handle, 1,
false, model_view_projection_matrix, 0);

GLES20.glVertexAttribPointer(texture_vertex_handle, 3,
GLES20.GL_FLOAT, false, 0, vertex_buffer);
GLES20.glEnableVertexAttribArray(texture_vertex_handle);

GLES20.glVertexAttribPointer(texture_texcoord_handle, 2,
GLES20.GL_FLOAT, false, 0, texcoord_buffer);
GLES20.glEnableVertexAttribArray(texture_texcoord_handle);

GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, texid);
GLES20.glUniform1i(texture_image_handle, 0);

GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);

GLES20.glUseProgram(0);
```

Alpha blending

- Multiple shader program
  - Simple shader
  - Texture shader

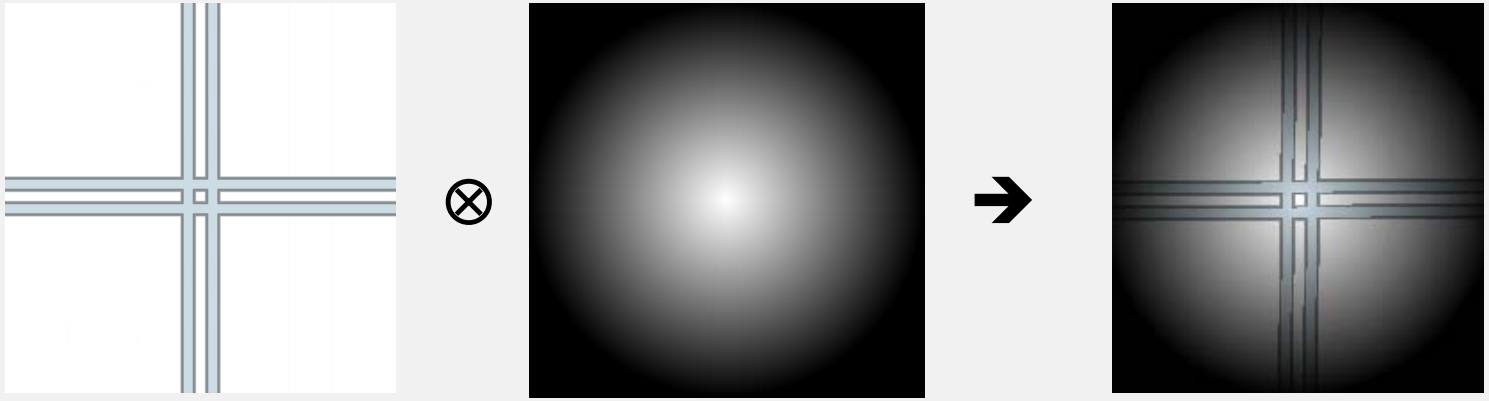
- Alpha blending
  - Using image alpha bits



Multi-Texturing with Programmable Rendering Pipeline

Multi-Texturing

- Texture blending
  - To generate a new texture image by blending given two or more textures
  - Very common operation in fragment shaders
    - ex) Precomputed lighting, normal maps

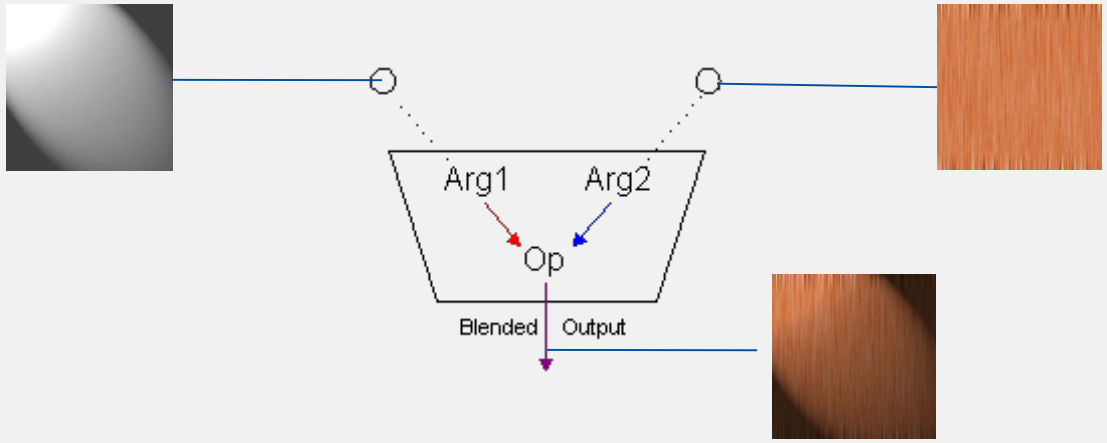


Multi-Texturing

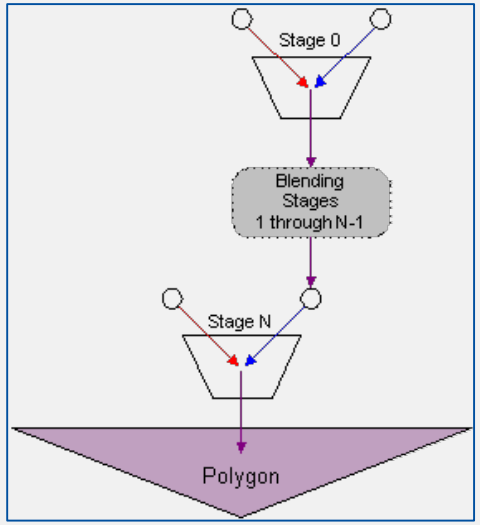
- Quake 2
  - http://www.bigpanda.com/trinity/article1.html

Multi-Texturing

- You can imagine a coffee machine to understand multi-texturing
  - In the fixed rendering pipeline of DirectX 9 and OpenGL



1-texture stage



2-texture stages

Multi-Texturing

Vertex / Fragment Shaders

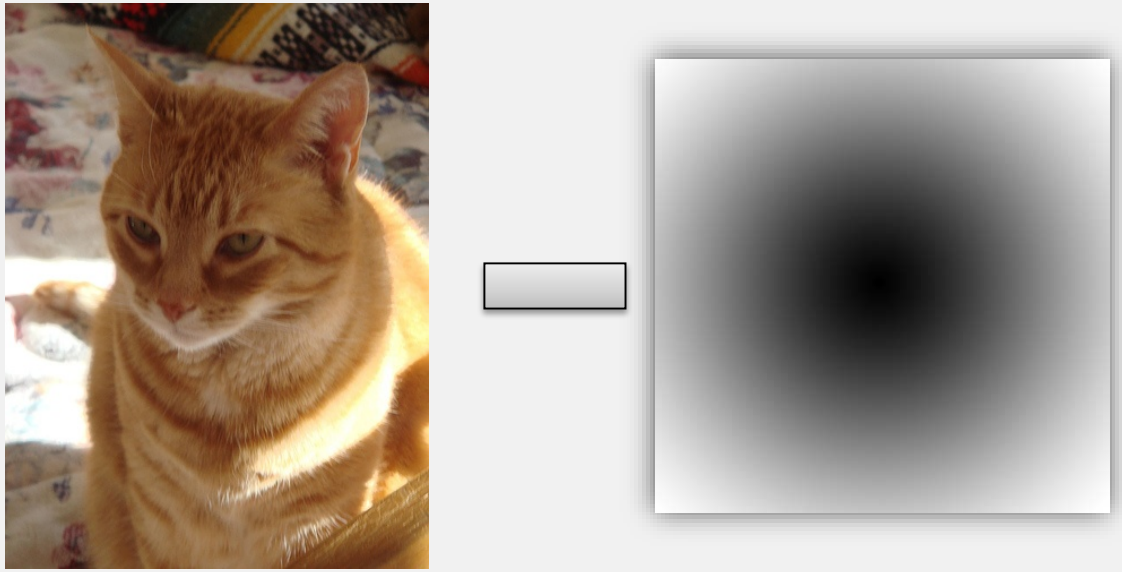
```
/// Multitexture Fragment Shader

precision mediump float;

varying vec2 v_texCoord;
uniform sampler2D s_baseMap;
uniform sampler2D s_lightMap;

void main()
{
  vec4 baseColor;
  vec4 lightColor;

  baseColor = texture2D(s_baseMap, v_texCoord);
  lightColor = texture2D(s_lightMap, v_texCoord);

  glFragColor = baseColor * (lightColor + 0.25);
}
```

OpenGL ES codes (C/C++)

```
GLuint  baseMapLoac, baseMapTexId;


// Bind the base map
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, baseMapTexId);

// Set the base map sampler to texture unit 0
glUniform1i(baseMapLoc, 0);


// Bind the light map
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, lightMapTexId);

// Set the base map sampler to texture unit 1
glUniform1i(lightMapLoc, 1);
```

Practice – Multi-Texturing

- Multi-texture
  - Activate texture units
  - Operation between textures



Multi-Texturing

Texture mapping

- Uniform / attribute
  - model-view-projection matrix
  - Vertex, texture coordinate
- Vertex shader
  - Transform vertex
  - Send texture coordinate
- Fragment shader
  - Access a texture using sampler
  - Mix teture colors

Vertex / Fragment Shaders

```
uniform mat4 mvp_matrix;

attribute vec4 a_vertex;
attribute vec2 a_texcoord;

varying vec2 v_texcoord;

void main() {
  v_texcoord = a_texcoord;
  gl_Position = mvp_matrix * a_vertex;
}
```

```
precision mediump float;

uniform sampler2D cat
uniform sampler2D gradient

varying vec2 v_texcoord;

void main() {
  vec4 cat_color = texture2D(cat, v_texcoord);
  vec4 gradient = texture2D(gradient, v_texcoord);
  gl_FragColor = cat_color - sqr_gradient;
}
```

Multi-Texturing

Texture initializion

1. Create texture object
2. Bind texture object
3. Send texture image to GPU memory
4. Specify texture parameter
   - Magnification, minifying function
   - Wrap
5. Generate mipmap

Android codes

```
if(texture_in){
  GLES20.glGenTextures(2, textures, 0);

  // first texture
  GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[0]);
  GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, first_bmp, 0);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                  GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                  GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                  GLES20.GL_TEXTURE_WRAP_S, GLES20.GL_CLAMP_TO_EDGE);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                  GLES20.GL_TEXTURE_WRAP_T, GLES20.GL_CLAMP_TO_EDGE);

  // second texture
  GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[1]);
  GLUtils.texImage2D(GLES20.GL_TEXTURE_2D, 0, second_bmp, 0);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                   GLES20.GL_TEXTURE_MIN_FILTER, GLES20.GL_LINEAR);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                   GLES20.GL_TEXTURE_MAG_FILTER, GLES20.GL_LINEAR);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                   GLES20.GL_TEXTURE_WRAP_S, GLES20.GL_CLAMP_TO_EDGE);
  GLES20.glTexParameterf(GLES20.GL_TEXTURE_2D,
                   GLES20.GL_TEXTURE_WRAP_T, GLES20.GL_CLAMP_TO_EDGE);

  GLES20.glGenerateMipmap(GLES20.GL_TEXTURE_2D);

  // remove bmp
  first_bmp.recycle();
  first_bmp = null;
  second_bmp.recycle();
  second_bmp = null;

  texture_in = false;
}
```

Texture Mapping with Sampler

On drawing

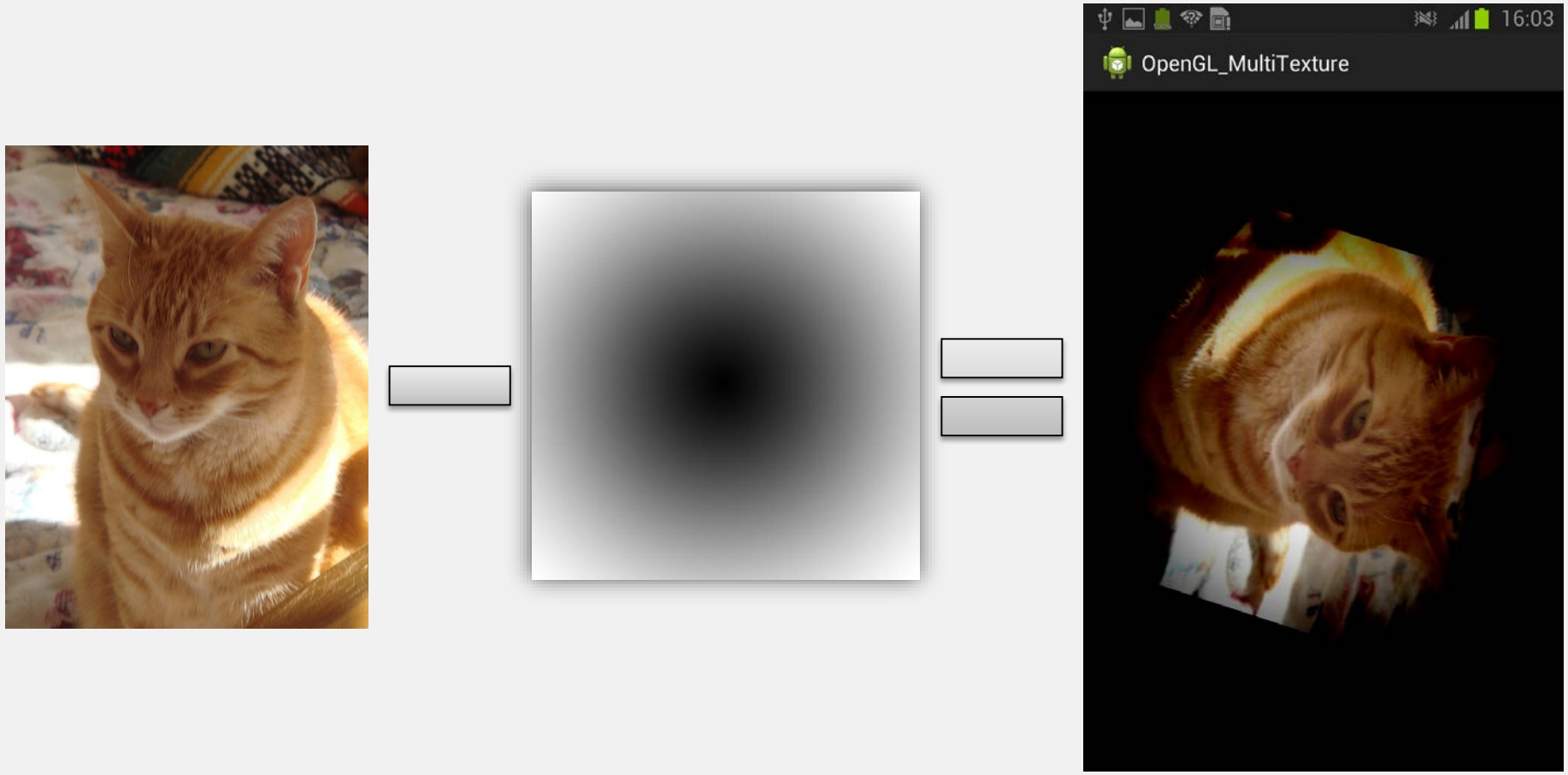
1. Activate texture unit
   - Texture0, texture1
2. Bind texture

3. Link texture unit with texture location
4. Draw object

Android codes

```
// draw
GLES20.glUseProgram(program);

GLES20.glUniformMatrix4fv(mvp_matrix_handle, 1, false,
                          projection_view_model_matrix, 0);
checkGlError("glUniformMatrix4fv mvp_matrix_handle");

GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
checkGlError("glActiveTexture GL_TEXTURE0");
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[0]);
checkGlError("glBindTexture textures[0]");

GLES20.glUniform1i(cat_handle, 0);
checkGlError("glUniform1i 0");

GLES20.glActiveTexture(GLES20.GL_TEXTURE1);
checkGlError("glActiveTexture GL_TEXTURE1");
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, textures[1]);
checkGlError("glBindTexture textures[0]");

GLES20.glUniform1i(gradient_handle, 1);
checkGlError("glUniform1i 1");

cube.draw(vertex_handle, texcoord_handle);

GLES20.glUseProgram(0);
checkGlError("glUseProgram 0");
```

Multi-Texturing

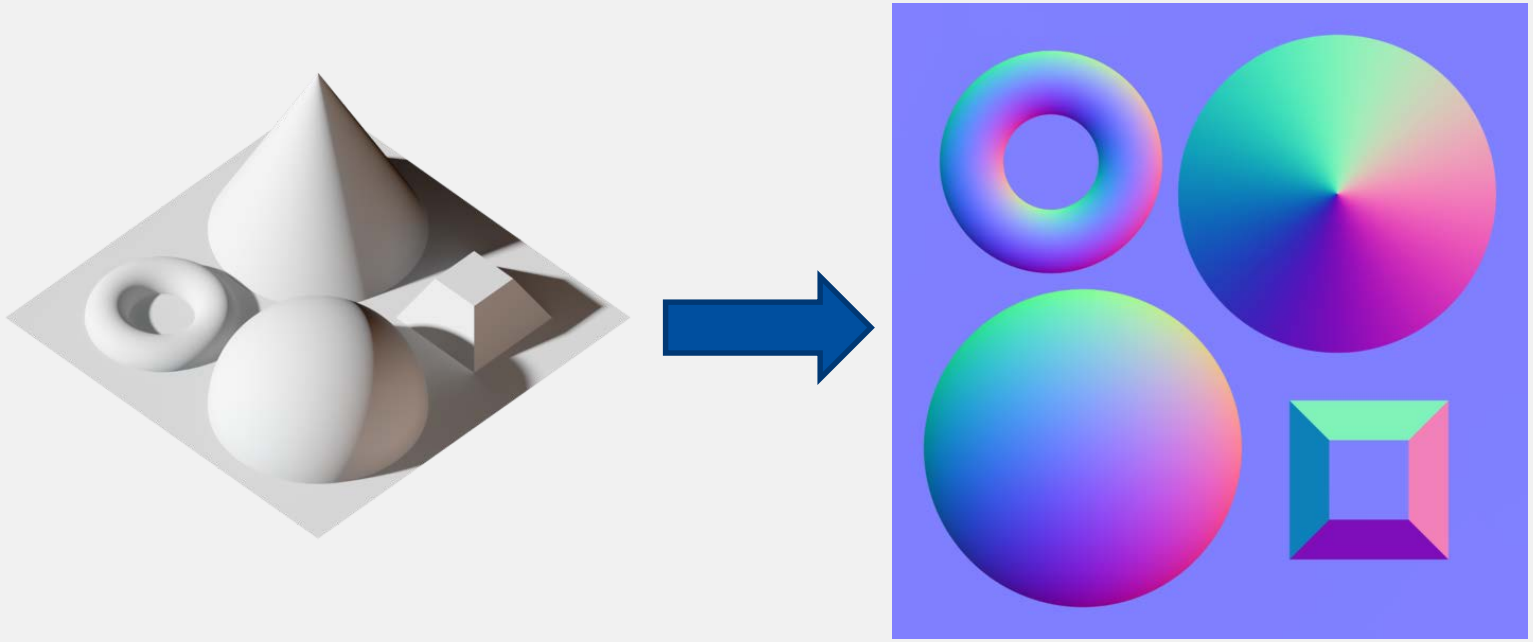


Normal Mapping

- Available w/ Pixel Shaders

Practice – Normal Mapping

- Normal map
  - Image
  - RGB as a normal vector



http://en.wikipedia.org/wiki/Normal_mapping

Normal Mapping

Vertex shader

```
uniform mat4 mvp_matrix;
uniform mat4 mv_matrix;

attribute vec4 a_vertex;
attribute vec3 a_texcoord;

varying vec3 v_vertex;
varying vec3 v_texcoord;

void main() {
 v_texcoord = a_texcoord;
  v_vertex = (mv_matrix * a_vertex).xyz;

  gl_Position = mvp_matrix * a_vertex;
}
```

Fragment shader

```
precision mediump float;

uniform mat3 normal_matrix;
uniform sampler2D normal_map

varying vec3 v_vertex;
varying vec3 v_texcoord;

vec3 light_position  = vec3(1.0, 1.0, 1.0);
vec4 light_diffuse    = vec4(1.0, 1.0, 1.0, 1.0);
vec4 light_specular  = vec4(1.0, 1.0, 1.0 1.0);
vec4 material_diffuse = vec4(0.5, 0.5, 0.5 1.0);
vec4 material_specular = vec4(1.0, 1.0, 1.0 1.0);
float material_shininess = 50.0;

vec4 directional_light(vec3 normal) {
  vec4 color = vec4(0.2,0.2,0.2,1);
  vec3 normal = normalize(normal_matrix * normal);
  vec3 light_vector = normalize(light_position);
  vec3 reflect_vector = reflect(-light_vector, normal);
  vec3 view_vector = normalize(-v_vertex);

  float ndotl = max(0.0, dot(normal, light_vector) );
  color += (ndotl * light_diffuse * material_diffuse);

  float rdotv = max(0.0, dot(reflect_vector, view_vector) );
  color += (pow(rdotv, material_shininess) * light_specular * material_specular);
  return color;
}

void main() {
  vec3 normal = texture2D(normal_map, v_texcoord).xyz;
  gl_FragColor = directional_light(normal);
}
```

Normal Mapping

Model

- Simple rectangle

- Add texture coordinate

Android codes

```
private float[] vertices = new float[] {
  -0.5f,  0.5f, 0.0f,  // left top
   0.5f,  0.5f, 0.0f,  // right top
  -0.5f, -0.5f, 0.0f,  // left bottom
   0.5f, -0.5f, 0.0f,  // right bottom
};

private float[] texcoords = new float[] {
  0.0f, 1.0f, // left top
  1.0f, 1.0f, // right top
  0.0f, 0.0f, // right bottom
  1.0f, 0.0f, // left bottom
};

private FloatBuffer vertex_buffer;
private FloatBuffer texcoord_buffer;
```

Texture Mapping with Sampler

On drawing

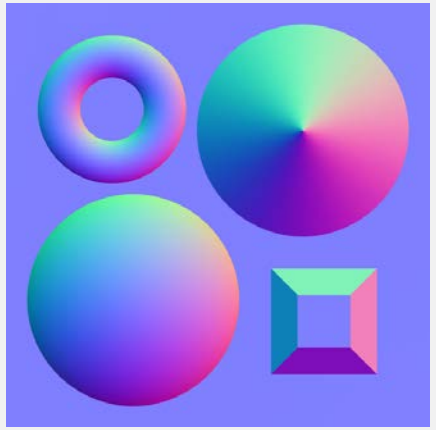
- Setting matrices
- Texture
  1. Activate texture unit
  2. Bind texture
  3. Link texture unit with texture location
- Simply draw a rectangle

Android codes

```
// draw
GLES20.glUseProgram(program);

GLES20.glUniformMatrix4fv(mvp_matrix_handle, 1, false,
model_view_projection_matrix, 0);
GLES20.glUniformMatrix4fv(mv_matrix_handle, 1, false,
model_view_matrix, 0);
GLES20.glUniformMatrix3fv(normal_matrix_handle, 1, false,
normal_matrix, 0);

GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, normal_map);
GLES20.glUniform1i(normal_map_handle, 0);

GLES20.glVertexAttribPointer(vertex_handle,
          3, GLES20.GL_FLOAT, false, 0, vertex_buffer);
GLES20.glEnableVertexAttribArray(vertex_handle);

GLES20.glVertexAttribPointer(texcoord_handle,
          2, GLES20.GL_FLOAT, false, 0, texcoord_buffer);
GLES20.glEnableVertexAttribArray(texcoord_handle);

GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);

GLES20.glUseProgram(0);
```

Normal Mapping

- Lighting with normal map



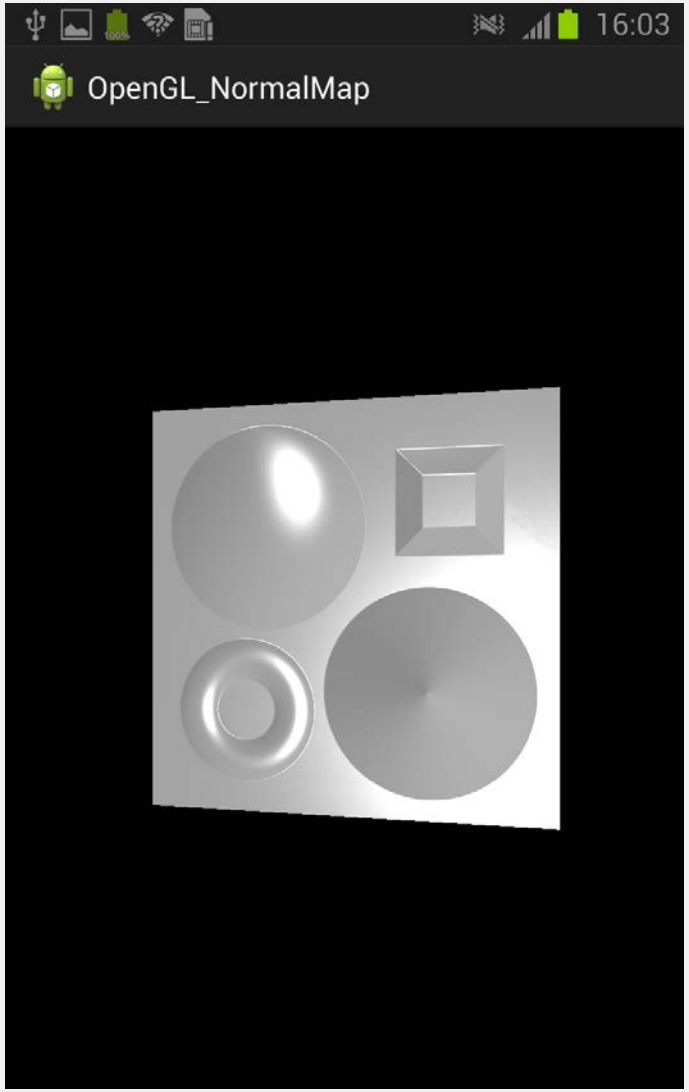


Image Processing

- Image contrast, brightness, blur, sharpness, saturation processing

Image Processing

- Convolution
  - Common image processing operation
  - Simple image filtering
  - Smoothing, edge detection, sharpening …

0	-1	0
-1	4	-1
0	-1	0

edge detection kernel

Image Processing

Vertex shader

```
uniform mat4 mvp_matrix;

attribute vec4 a_vertex;
attribute vec3 a_texcoord;

varying vec3 v_texcoord;

void main() {
  v_texcoord = a_texcoord;
  gl_Position = mvp_matrix * a_vertex;
}
```

Fragment shader

```
precision mediump float;

const int MAX_KERNEL_SIZE = 25;

uniform vec2 offset[MAX_KERNEL_SIZE];
uniform float kernel[MAX_KERNEL_SIZE];
uniform int kernel_size;
uniform float denominator;
uniform sampler2D image;

varying vec3 v_texcoord;

void main() {
  vec4 sum = vec4(0.0);
  for(int i=0; i<kernel_size; ++i){
     vec4 tmp = texture2D(image, v_texcoord.st + offset[i]);
    sum += kernel[i] * tmp;
  }
  gl_FragColor = sum*denominator;
}
```

Image Processing

Kernel data

- Offset
  - Pixel locations for convolution operation
  - Need to be resized
- Kernel
  - Each pixel weight

Android codes

```
private float[] offset = {
  -1, 1,  0, 1,  1, 1,
  -1, 0,  0, 0,  1, 0,
  -1,-1,  0,-1,  1,-1,
};
private float[] kernel = {
   0, -1,  0,
  -1,  4, -1,
   0, -1,  0
};

private int kernel_size = 9;
private float denominator = 10.0f;
```

```
for(int i=0; i<kernel_size; ++i){
  offset[i*2 + 0] = offset[i*2 + 0] / (float)width;
  offset[i*2 + 1] = offset[i*2 + 1] / (float)height;
}
```

Image Processing

Fit image to view

- Fit rectangle to view

```
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {
  GLES20.glViewport(0, 0, width, height);
  Matrix.orthoM(projection_matrix, 0,
                -1, 1, -1, 1, 1, 10);

  for(int i=0; i<kernel_size; ++i){
    offset[i*2 + 0] = offset[i*2 + 0] / (float)width;
    offset[i*2 + 1] = offset[i*2 + 1] / (float)height;
  }
}
```

```
private static String TAG = "GLRenderer";

private float[] vertices = new float[] {
-1.0f,  1.0f,  0.0f,  // left top
 1.0f,  1.0f,  0.0f,  // right top
-1.0f, -1.0f,  0.0f,  // left bottom
 1.0f, -1.0f,  0.0f,  // right bottom
};

private float[] texcoords = new float[] {
0.0f, 0.0f,  // left top
1.0f, 0.0f,  // right top
0.0f, 1.0f,  // left bottom
1.0f, 1.0f,  // right bottom
};

private FloatBuffer vertex_buffer;
private FloatBuffer texcoord_buffer;
```

Image Processing

On drawing

- Setting matrices
- Setting kernel data
  - Offset, kernel value, kernel size
- Texture
  1. Activate texture unit
  2. Bind texture
  3. Link texture unit with texture location
- draw a rectangle

Android codes

```
// draw
GLES20.glUseProgram(program);

GLES20.glUniformMatrix4fv(mvp_matrix_handle, 1, false,
model_view_projection_matrix, 0);

GLES20.glUniform2fv(offset_handle, kernel_size, offset, 0);
GLES20.glUniform1fv(kernel_handle, kernel_size, kernel, 0);
GLES20.glUniform1i(kernel_size_handle, kernel_size);
GLES20.glUniform1f(denominator_handle, denominator);

GLES20.glActiveTexture(GLES20.GL_TEXTURE0);
GLES20.glBindTexture(GLES20.GL_TEXTURE_2D, tex_id);

GLES20.glUniform1i(image_handle, 0);

GLES20.glVertexAttribPointer(vertex_handle, 3,
GLES20.GL_FLOAT, false, 0, vertex_buffer);
GLES20.glEnableVertexAttribArray(vertex_handle);

GLES20.glVertexAttribPointer(texcoord_handle, 2,
GLES20.GL_FLOAT, false, 0, texcoord_buffer);
GLES20.glEnableVertexAttribArray(texcoord_handle);

GLES20.glDrawArrays(GLES20.GL_TRIANGLE_STRIP, 0, 4);

GLES20.glUseProgram(0);
```

Image Processing

- Highlighted edges