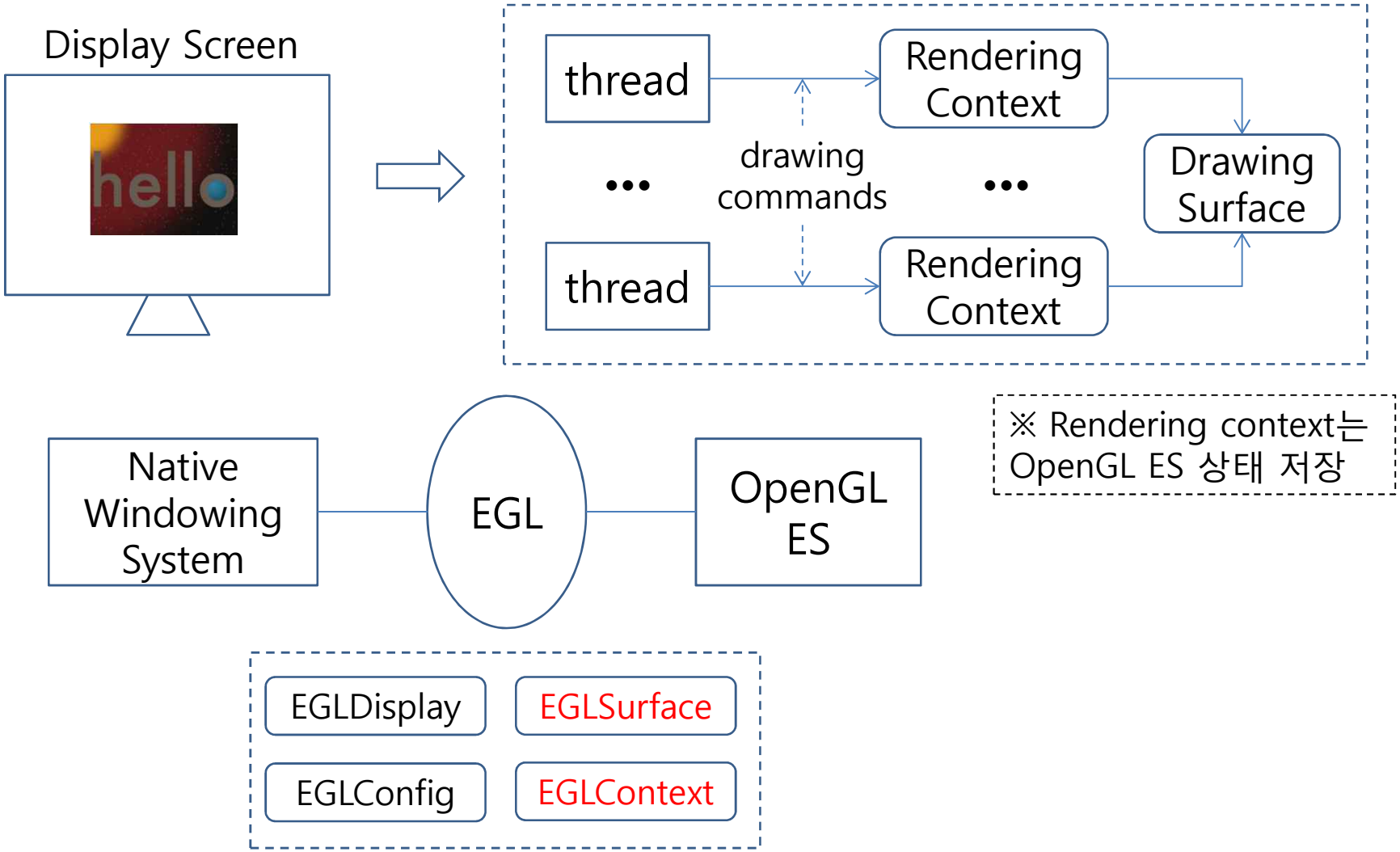
Shading Language in Android

우덕균, 이경민

순서

- OpenGL ES Shading Language and Shader
  - EGL & OpenGL ES 3.0 Graphics Pipeline
  - OpenGL ES Shading Language
  - Compiling and Linking Shader Source
  - Vertex Shader
  - Fragment Shader
  - Using Textures in a Shader
- Patterns of Shaders in Android
  - Simple Pattern in Screen Recorder Command
  - Extensible Pattern in Mobile Filter Framework
  - Flexible Pattern in SurfaceFlinger and HWUI

EGL
(Embedded-System Graphics Library)



EGL Functions (1)

- Communicating with the Windowing System
  - Native windowing system과 OpenGL ES 응용과의 communication channel 생성
  - EGL 내부 자료 구조 초기화

```
EGLDisplay display = eglGetDisplay(EGL_DEFAULT_DISPLAY);

EGLint major, minor;
eglInitialize(display, &major, &minor);
```

- Determining the Available Surface Configurations
  - 모든 surface configuration을 가져와 응용이 surface 결정
    - eglGetConfigs(), eglGetConfigAttrib()
  - EGL system이 가장 적합한 surface configuration을 결정
    - eglChooseConfig()

```
const EGLint configAttribs[] =
  {
    EGL_RENDER_TYPE, EGL_WINDOW_BIT,
    EGL_RED_SIZE, 8,
    EGL_GREEN_SIZE, 8,
    EGL_BLUE_SIZE, 8,
    EGL_DEPTH_SIZE, 24,
    EGL_NONE
  };

EGLConfig config;
EGLint numConfigs;
eglChooseConfig(display, configAttribs, &config, 1, &numConfigs);
```

EGL Functions (2)

- Creating an On-Screen Rendering Area: The EGL Window

```
EGLSurface window =
  eglCreateWindowSurface(display, config, nativeWindow, NULL);
```

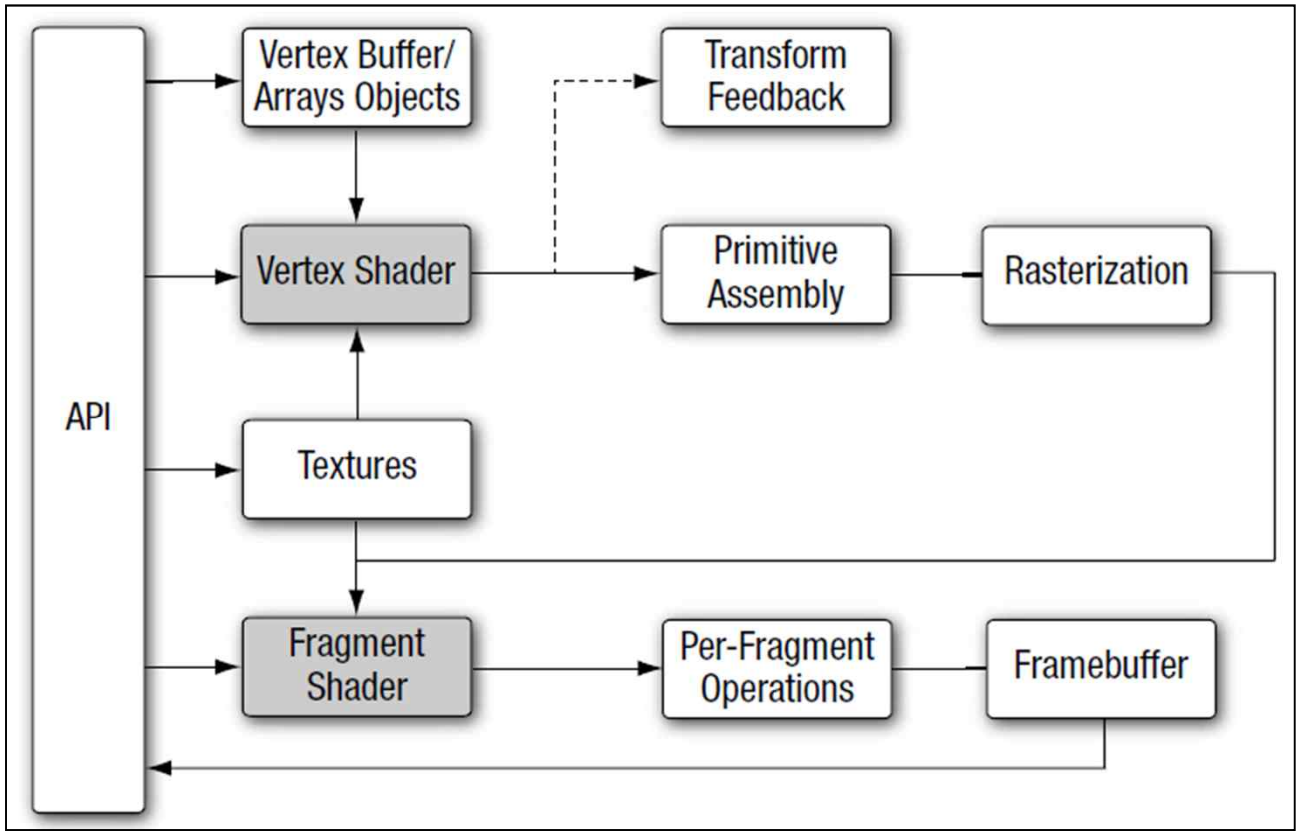
- Creating an Off-Screen Rendering Area: EGL Pbuffers
  - pbuffer (short for pixel buffer) : nonvisible off-screen surfaces
  - eglCreatePbufferSurface()

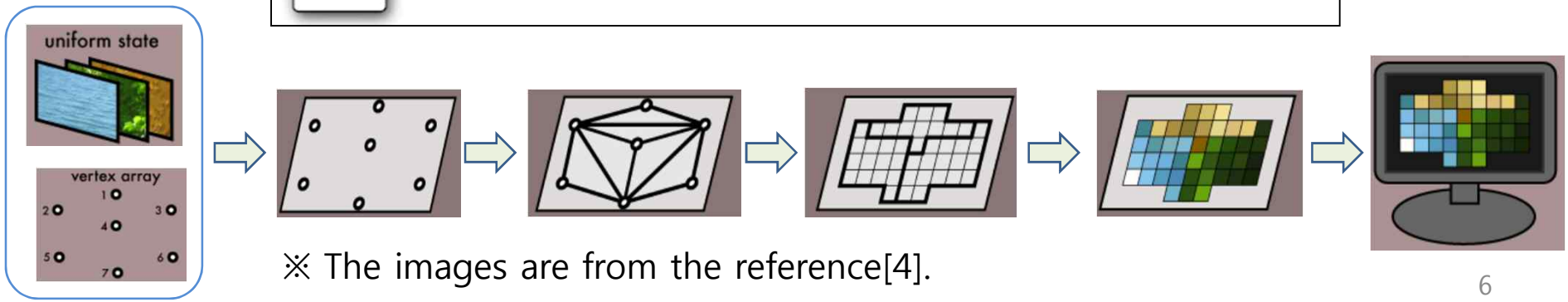
- Creating a Rendering Context and Making an EGL Context Current

```
const EGLint contextAttribs[] =
  {
    EGL_CONTEXT_CLIENT_VERSION, 3,
    EGL_NONE
  };

EGLContext context =
  eglCreateContext(display, config, EGL_NO_CONTEXT, contextAttribs);

eglMakeCurrent(display, window, window, context);
```

- Displaying the Framebuffer on the Screen
  - eglSwapBuffers(display, surface)

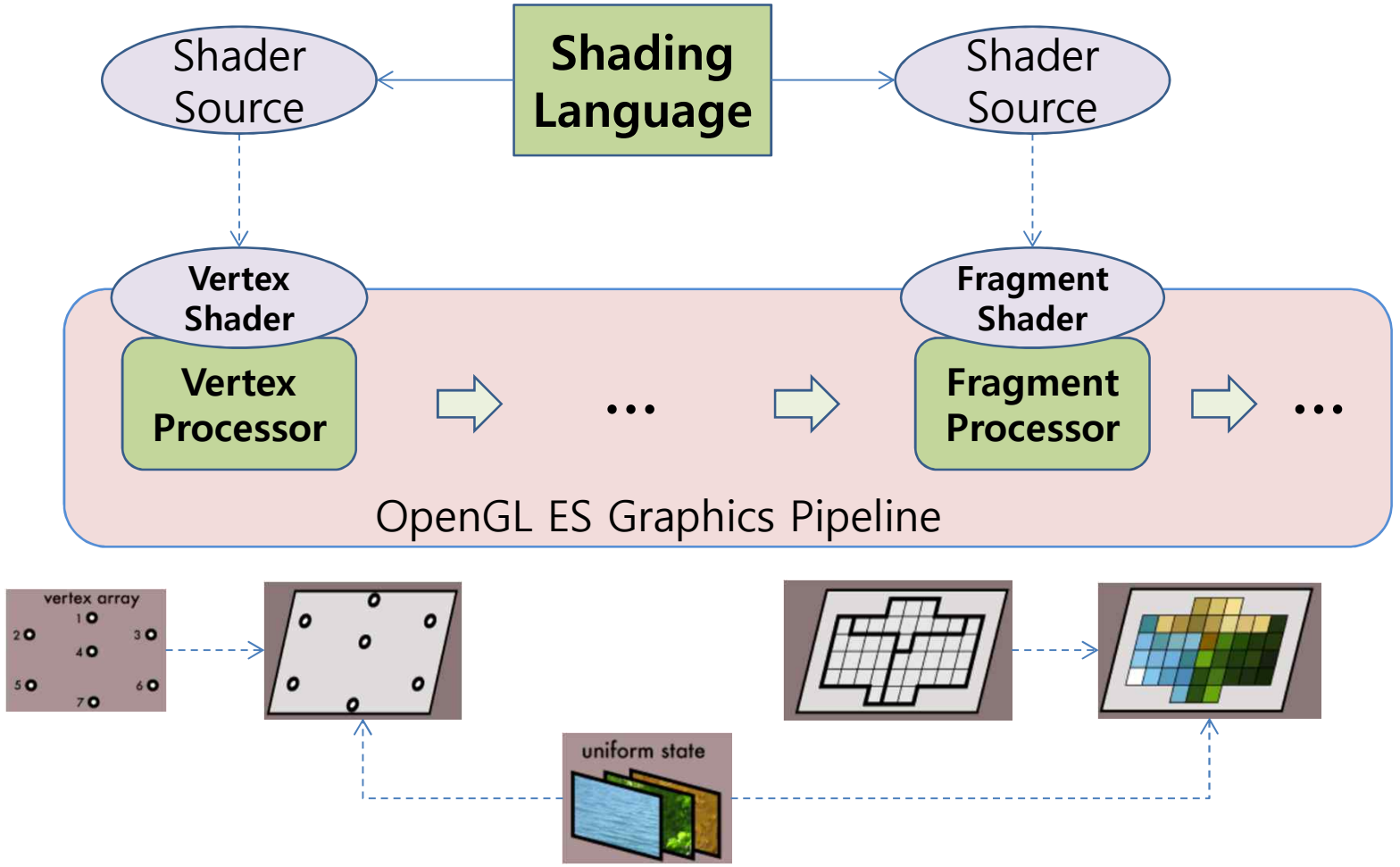
OpenGL ES 3.0 Graphics Pipeline (1)





※ The images are from the reference[4].

OpenGL ES 3.0 Graphics Pipeline (2)



OpenGL ES Shading Language

- C-style's programming language to give data processing commands to OpenGL ES Pipeline
  - Similar to OpenGL Shading Language (GLSL)
  - Supported from OpenGL ES 2.0 (fixed-function pipeline is eliminated)

- Language Features
  - Data Values, Types, and Variables
  - Data Operations
  - Conditions and Loops, Functions
  - Preprocessor Directives & Macros

- Differences from C Language
  - No Pointer
  - Added Vector and Matrix
  - Interface to Shader

```
#version 300 es

uniform float timer;

layout(location = 0) in vec3 position;

out vec2 texcoord;
out float fade_factor;

void main()
{
  gl_Position = vec4(position, 1.0);
  texcoord = position.xy * vec2(0.5) + vec2(0.5);
  fade_factor = sin(timer) * 0.5 + 0.5;
}
```

< Vertex Shader Source >

Data Values & Types

- Scalars
  - boolean (true, false) : bool
  - integer number : int, uint
  - floating-point number : float
- Vectors
  - floating-point : vec2, vec3, vec4
  - integer : ivec2, ivec3, ivec3
  - boolean : bvec2, bvec3, bvec4

to access vector components
- x : the first element
- y : the second element
- z : the third element
- w : the fourth element
- (ex) v.x, v.xyzw, v.yw

- Matrices (element is a floating-point number) // column-major order
  - 2x2 matrix : mat2
  - 3x3 matrix : mat3
  - 4x4 matrix : mat4
  - *mxn* matrix (m : column, n : row) : mat*mxn*

```
mat3 m;
m[0] // the first column
m[1] // the second column
m[2] // the third column
```

- Samplers (handles for texture lookups) // function pointer
  - access a one-dimensional textures : sampler1D
  - access a two-dimensional textures : sampler2D
  - …

Data Values & Types, Variables (1)

- Structures & Arrays
  - similar to C, bit-fields are not supported
- Declarations, Initializers, and Constructors
  - similar to C++

```
vec3 v1;
v1 = vec3(1.0, 2.0, 3.0);

vec3 v2 = vec3(4.0, 5.0, 6.0);
v1[2] = v2[2];

vec3 v3 = vec3(7.0);

mat2 m1 = mat2(1.0, 2.0, 3.0, 4.0);
mat2 m2 = mat2(v1.xy, v2.yz);
```

$$m1 = \begin{bmatrix} 1.0 & 3.0 \\ 2.0 & 4.0 \end{bmatrix} \quad m2 = \begin{bmatrix} 1.0 & 5.0 \\ 2.0 & 6.0 \end{bmatrix}$$

```
vec4 v4 = vec4(1.0, 2.0, 3.0, 4.0);
v4.x // 1.0
v4.xy // (1.0, 2.0)
v4.wzxy // (4.0, 3.0, 1.0. 2.0)
v4.xx // (1.0, 1.0)
```

```
struct light {
  vec3 position;
  vec3 color;
} a;
light b;
```

```
vec4 points[];
points[2] = vec4(1.0);
points[7] = vec4(2.0);
```

```
float a[4] = float[4](1.0, 1.0, 1.0, 1.0);
```

Data Values & Types, Variables (2)

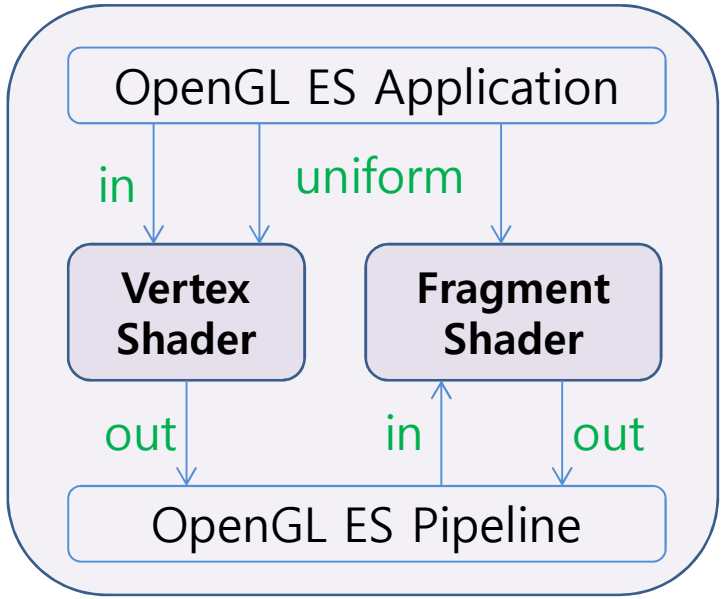
- Type Conversions

```
float f = 2.3;
bool b = bool(f);
```

```
bool b = true;
float f = float(3);
float g = float(b);
vec4 v = vec4(2);
```

- Qualifiers and Interface to a Shader
  - uniform : 거의 변하지 않는 데이터, shader내에서 writing 안됨
  - in, out : shader 마다 의미하는 바가 다름
  - const : shader내에서 사용되는 상수 값

```
in float Temperature;
const int NumLights = 3;
uniform vec4 lpos[NumLights];
```



Data Operations

Operator	Description
()	Parenthetical grouping
[ ] () . ++ --	Index Function call and constructor Member selection and swizzle Postfix increment/decrement
++ -- + - ~ !	Prefix increment/decrement Unary
* / %	Multiplicative
+ −	Additive
<< >>	Bit-wise shift
< > <= >=	Relational
== !=	Equality
&	Bit-wise and
^	Bit-wise exclusive or
|	Bit-wise inclusive or
&&	Logical and

Operator	Description
^^	Logical exclusive or
||	Logical inclusive or
?:	Selection
= += -= *= /= %= <<= >>= &= ^= |=	Assignment Arithmetic assignment
,	Sequence

※ The image is from the reference[2].

Examples of Data Operations

```
vec4 v = vec4(1.0, 2.0, 3.0, 4.0);
float f = v[2];
```

```
mat4 m = mat4(3.0);
vec4 v;
v = m[1]; // (0.0, 3.0, 0.0, 0.0)
```

```
vec3 v, u, w;
float f;
v = u + f; // v.x = u.x + f, ...
w = v + u; // w.x = v.x + u.x, ...
```

```
vec2 v, u;
mat2 m, n;
v * u;
v * m;
m * v;
m * n;
```

v = (2.0, 3.0)
u = (4.0, 5.0)

$$m = \begin{bmatrix} 1.0 & 3.0 \\ 2.0 & 4.0 \end{bmatrix}$$

m[0] = (1.0, 2.0)
m[1] = (3.0, 4.0)

$$n = \begin{bmatrix} 5.0 & 7.0 \\ 6.0 & 8.0 \end{bmatrix}$$

v * u = (2.0*4.0, 3.0*5.0)

$$v * m = \begin{bmatrix} 2.0 * 1.0 + 3.0 * 2.0 \\ 2.0 * 3.0 + 3.0 * 4.0 \end{bmatrix}$$

$$m * v = \begin{bmatrix} 1.0 * 2.0 + 3.0 * 3.0 \\ 2.0 * 2.0 + 4.0 * 3.0 \end{bmatrix}$$

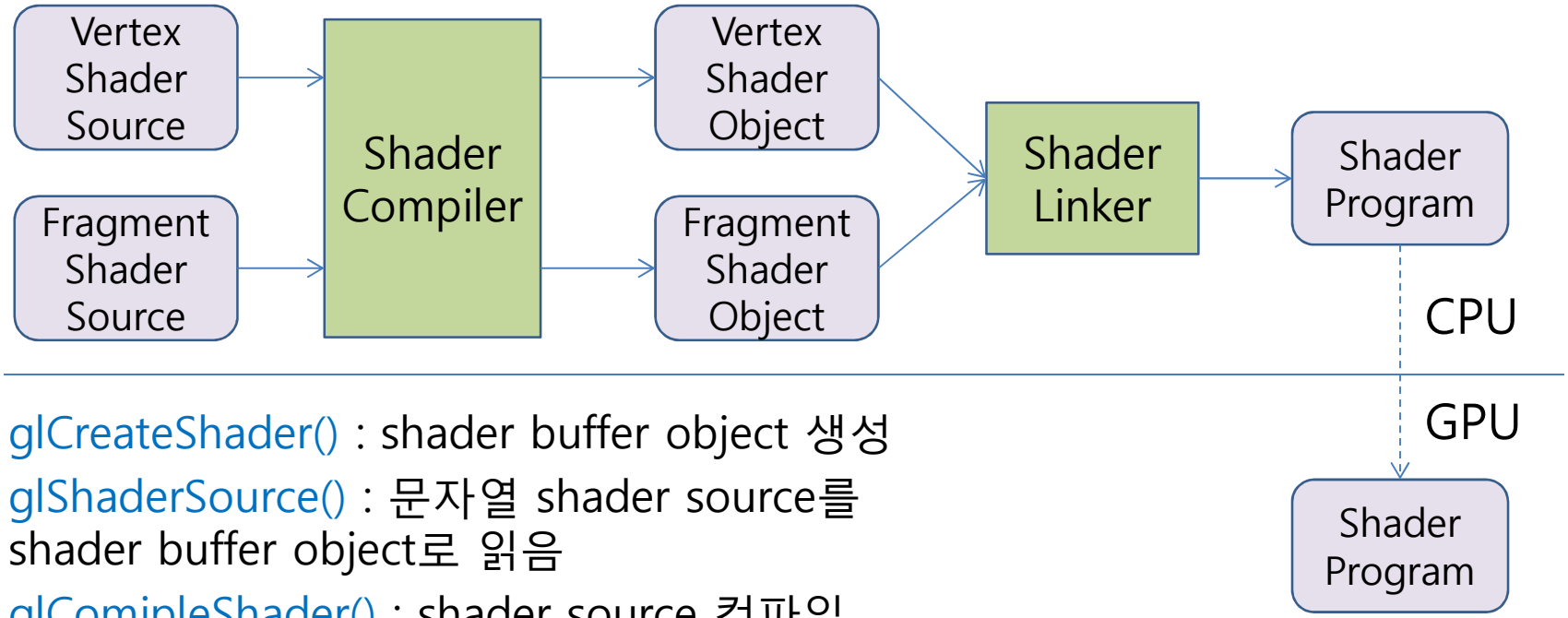
$$m * n = \begin{bmatrix} 1.0 * 5.0 + 3.0 * 6.0 & 1.0 * 7.0 + 3.0 * 8.0 \\ 2.0 * 5.0 + 4.0 * 6.0 & 2.0 * 7.0 + 4.0 * 8.0 \end{bmatrix}$$

Conditions and Loops, Functions

- Conditions (similar to C++)
  - if, if-else, ? :
  - a variable cannot be declared in the if statement
  - discard : prevents a fragment from updating the framebuffer
- Loops (similar to C++)
  - for, while, do-while
  - break, continue

- Functions (similar to C++)
  - main function
  - qualifiers for formal parameters
    - in : copy in but don't copy back out
    - out : only copy out
    - inout : copy in and copy out
    - const : no writable
  - built-in functions : sin(), cos(), …

Compiling and Linking Shader Source (in Runtime)



- glCreateShader() : shader buffer object 생성
- glShaderSource() : 문자열 shader source를 shader buffer object로 읽음
- glComipleShader() : shader source 컴파일
- glGetShaderiv() : 컴파일 결과 확인
- glCreateProgram() : shader program object 생성
- glAttachShader() : shader buffer object를 shader program object에 연결
- glLinkProgram() : 연결된 shader object들을 링킹하여 shader program 생성
- glGetProgramiv() : 링킹 결과 확인
- glUseProgram() : shader program을 GPU에 보내 pipeline에서 사용되도록 함

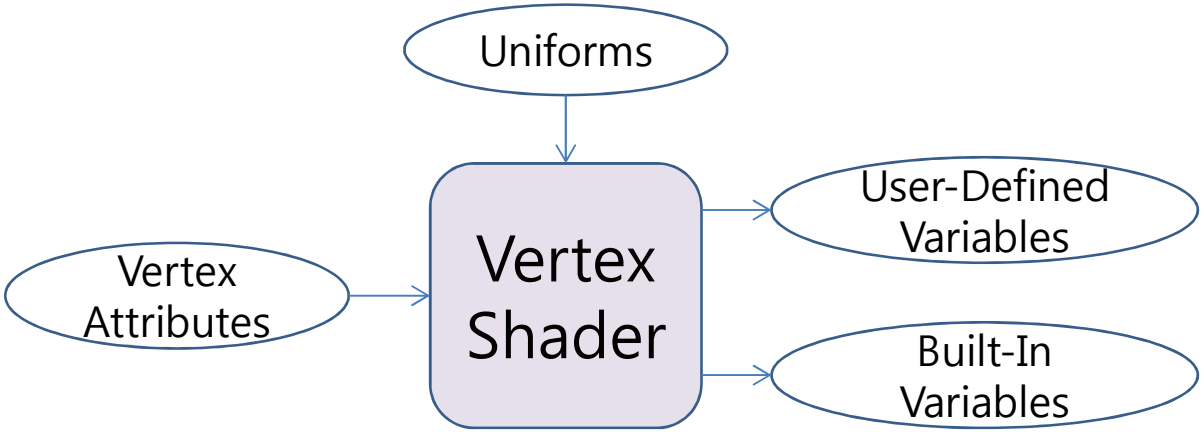
Example of Compiling and Linking

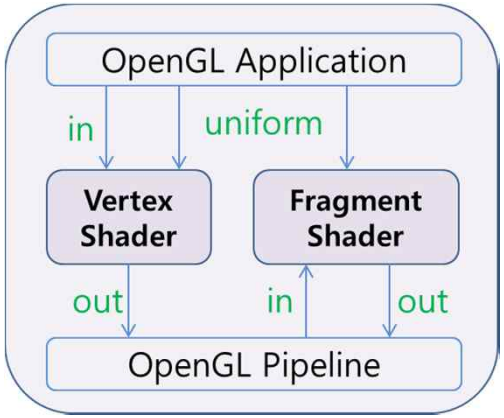
```
void make_shaders()
{
  v_shader = make_shader(GL_VERTEX_SHADER
  f_shader = make_shader(GL_FRAGMENT_SHAD

  program = make_program(v_shader, f_shac

  glUseProgram(program);
}
```

```
GLuint make_shader(GLenum type, const ch
{
  GLint length;
  GLchar *source = file_contents(filenam
  GLuint shader;
  GLint shader_ok;

  if (!source)
    return 0;

  shader = glCreateShader(type);
  glShaderSource(shader, 1, (const GLchar**)&source, &length);
  free(source);
  glCompileShader(shader);

  glGetShaderiv(shader, GL_COMPILE_STATUS, &shader_ok);
  if (!shader_ok) {
    fprintf(stderr, "Failed to compile %s:\n", filename);
    glDeleteShader(shader);
    return 0;
  }
  return shader;
}
```

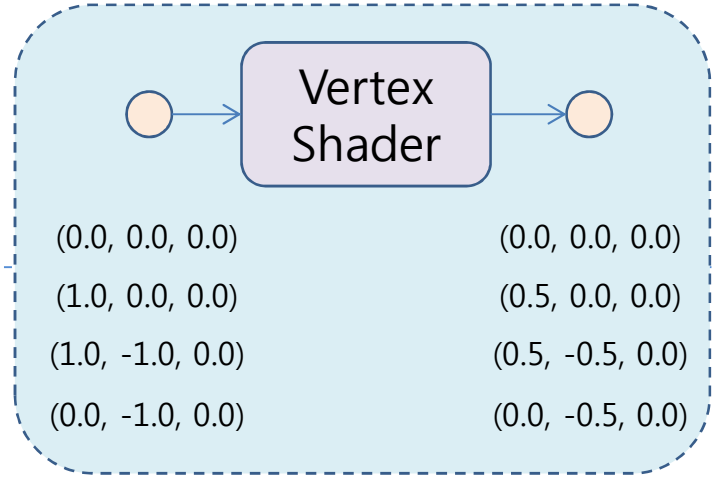
```
GLuint make_program(GLuint vertex_shader, GLuint fragment_shader)
{
  GLint program_ok;

  GLuint program = glCreateProgram();

  glAttachShader(program, vertex_shader);
  glAttachShader(program, fragment_shader);
  glLinkProgram(program);

  glGetProgramiv(program, GL_LINK_STATUS, &program_ok);
  if (!program_ok) {
    fprintf(stderr, "Failed to link shader program:\n");
    glDeleteProgram(program);
    return 0;
  }
  return program;
}
```

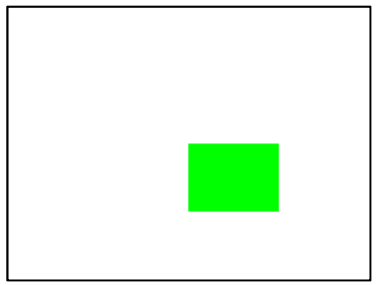
Vertex Shader





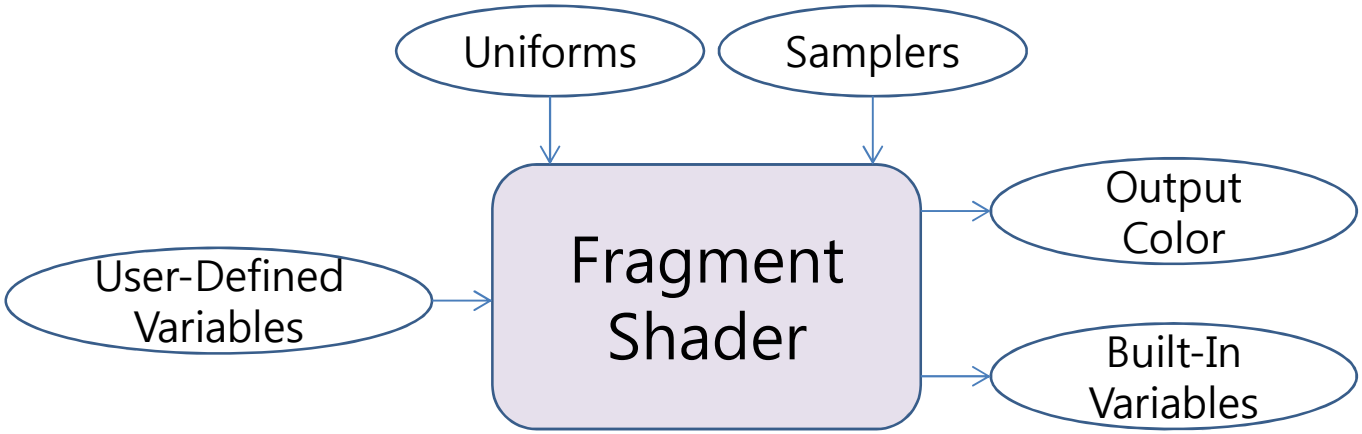

```
GLfloat vertices[][3] = {
  { 0.0,  0.0, 0.0 },
  { 1.0,  0.0, 0.0 },
  { 1.0, -1.0, 0.0 },
  { 0.0, -1.0, 0.0 },
};
```

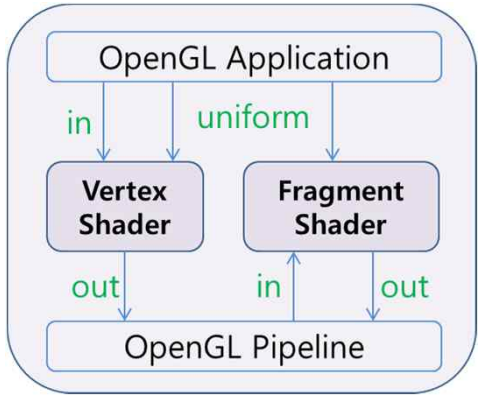




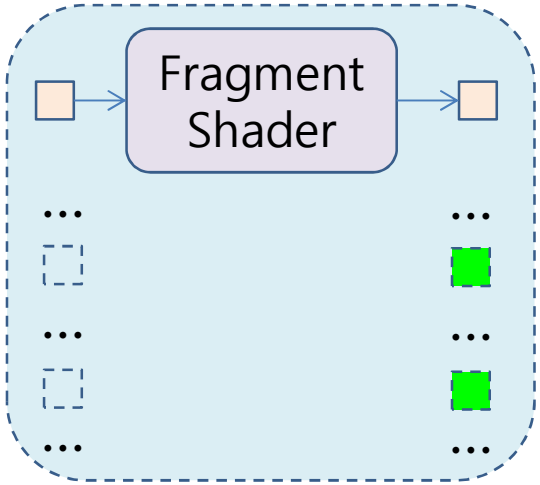
```
#version 300 es

layout(location = 0)
  in vec3 position;

void main()
{
  gl_Position = vec4(position.x*0.5, position.y*0.5, position.z, 1.0);
}
```

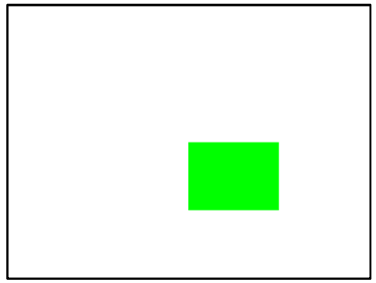
Fragment Shader



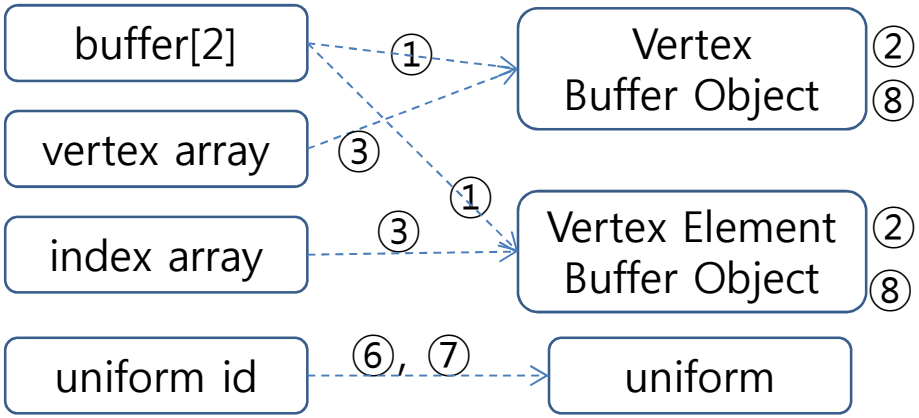



```
out vec4 FragColor;

void main()
{
    FragColor =
      vec4(0.0, 1.0, 0.0, 1.0);
}
```





Vertex Arrays, Buffer Objects, and Vertex Attributes

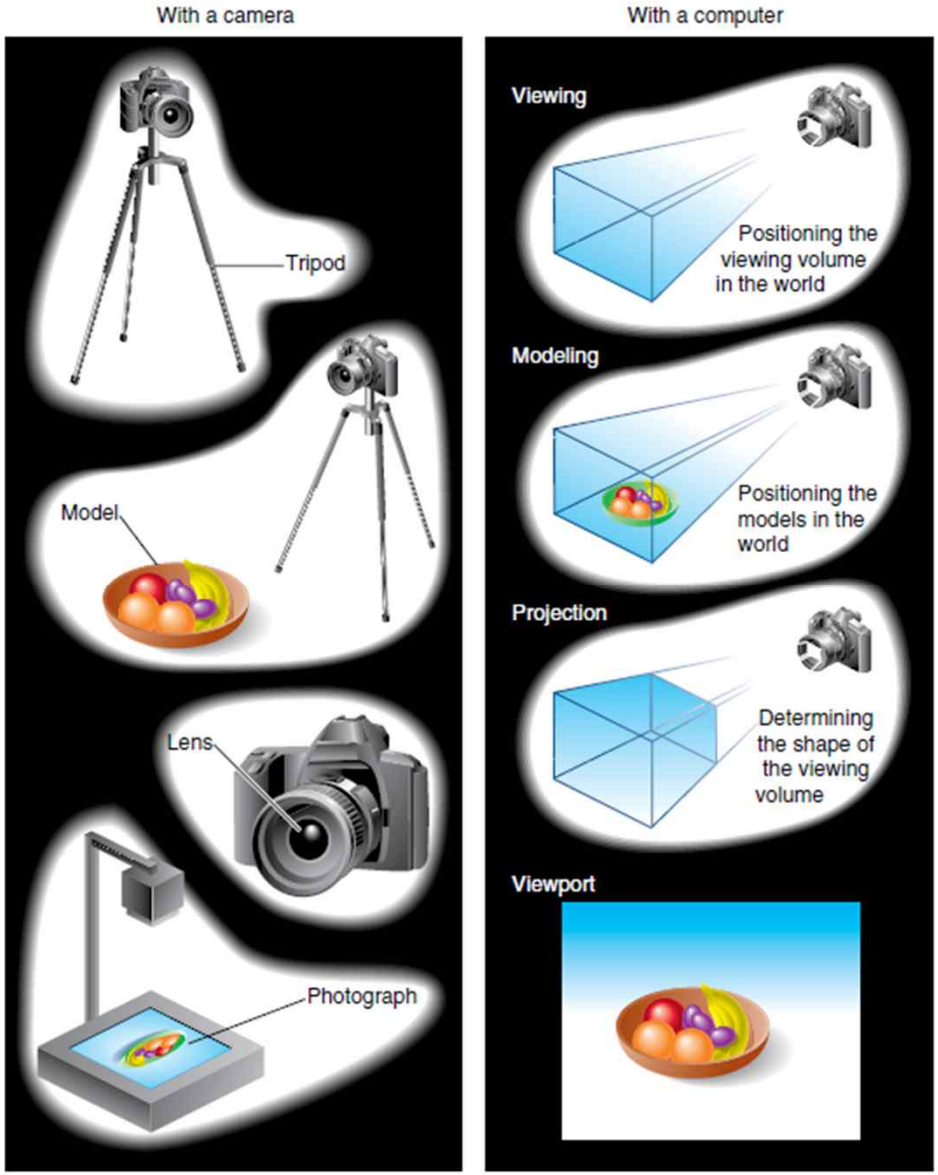



```
layout (location = 0) in vec3 position;
uniform float gFactor;
```

Vertex Shader

① glGenBuffers() : buffer object들 생성
② glBindBuffer() : buffer object 타입 결정 (array_buffer, element_array_buffer, ...)
③ glBufferData() : array data를 buffer object로 복사
④ glDrawArrays() : buffer object의 vertex array data에 대하여 rendering
⑤ glDrawElements() : vertex element array가 가리키는 data에 대하여 rendering
⑥ glGetUniformLocation() : shader program의 uniform 변수 위치 가져옴
⑦ glUniform1f() : shader program의 uniform 변수에 값을 할당
⑧ glVertexAttribPointer() : vertex attribute와 관련된 vertex buffer object 연결
⑨ glEnableVertexAttribArray() : vertex attribute 사용
⑩ glDisableVertexAttribArray() : vertex attribute 사용하지 않음

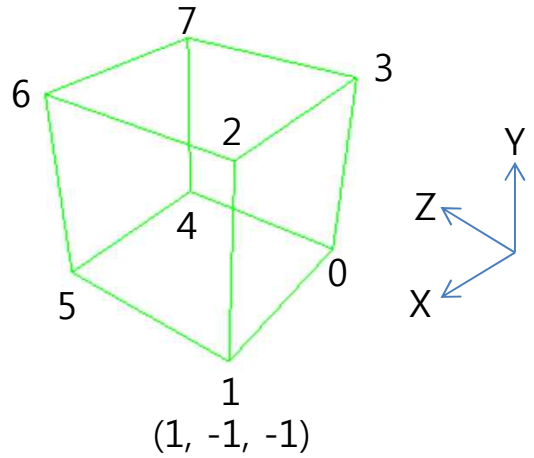
Viewing, Modeling, and Projection



※ The image is from the reference[1].

Drawing a Cube (1)

```
// [0] : array buffer, [1] : element buffer
GLuint buffers[NUM_BUFFERS];

GLint uniform_mvp;

glm::mat4 MVP;

GLfloat vertices[][3] = {
  {-1.0, -1.0, -1.0},
  { 1.0, -1.0, -1.0},
  { 1.0,  1.0, -1.0},
  {-1.0,  1.0, -1.0},

  {-1.0, -1.0,  1.0},
  { 1.0, -1.0,  1.0},
  { 1.0,  1.0,  1.0},
  {-1.0,  1.0,  1.0}
};

GLubyte indices[] = {
  0, 1, 1, 2, 2, 3, 3, 0,
  2, 3, 3, 7, 7, 6, 6, 2,
  4, 5, 5, 6, 6, 7, 7, 4,
  0, 1, 1, 5, 5, 4, 4, 0,
  1, 2, 2, 6, 6, 5, 5, 1,
  0, 3, 3, 7, 7, 4, 4, 0
};
```



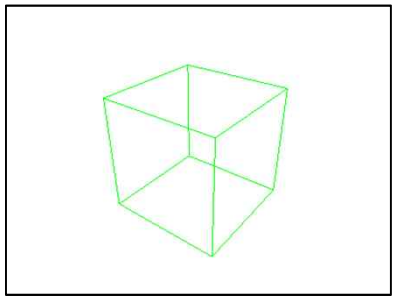

```
void make_mvp()
{
  glm::mat4 Projection =
    glm::perspective(45.0f, 4.0f / 3.0f, 0.1f, 100.0f);

  glm::mat4 View =
    glm::lookAt(glm::vec3(4, 3, -3),
                glm::vec3(0, 0, 0),
                glm::vec3(0, 1, 0));

  glm::mat4 Model = glm::mat4(1.0);

  MVP = Projection * View * Model;

  uniform_mvp = glGetUniformLocation(program, "MVP");
}
```

< OpenGL ES Application >

Drawing a Cube (2)

```
void render(ESContext *esContext)
{
  glClear(GL_COLOR_BUFFER_BIT);

  glUniformMatrix4fv(uniform_mvp, 1, GL_FALSE, &MVP[0][0]);

  glBindBuffer(GL_ARRAY_BUFFER, buffers[0]);

  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(GLfloat)*3, 0);

  glEnableVertexAttribArray(0);

  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[1]);

  glDrawElements(GL_LINES, 48, GL_UNSIGNED_BYTE, (GLvoid*)0);
}
```

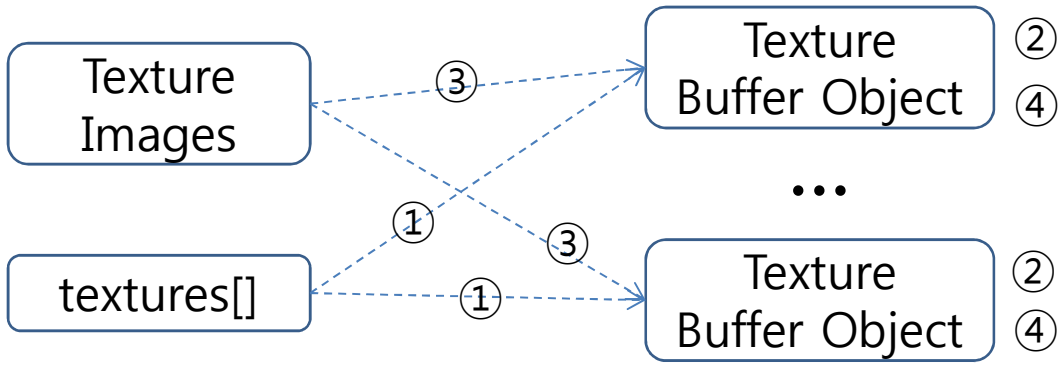
< OpenGL ES Application >



```
layout(location = 0) in vec3 position;

uniform mat4 MVP;

void main()
{
  gl_Position = MVP * vec4(position, 1.0);
}
```

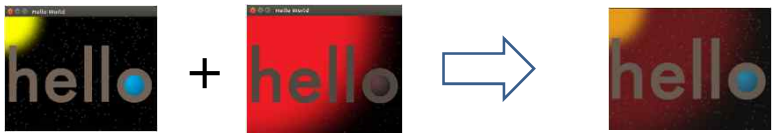
< Vertex Shader >

Using Textures in a Shader



① glGenTextures() : texture buffer object들 생성
② glBindTexture() : texture buffer object 타입 결정 (2D, 3D, …)
③ glTexImage2D() : 2D 이미지를 texture buffer object로 로딩
④ glTexParameteri() : texture의 filtering mode 설정
⑤ glActiveTexture() : 현재의 texture unit 설정 (0, 1, …), 다음에 glBindTexture()를 호출하여 해당 texture unit의 buffer object binding
⑥ texture() : fragment shader에서 사용되는 함수로 해당 fragment의 texture data (color 값) 가져옴

Fading a Textured Image (1)




```
GLuint make_texture(const char *filename)
{
  int width, height;
  void *pixels = read_tga(filename, &width, &height);
  GLuint texture;

  glGenTextures(1, &texture);
  glBindTexture(GL_TEXTURE_2D, texture);

  glTexImage2D
    (
     GL_TEXTURE_2D, 0,            /* target, level */
     GL_RGB8,                     /* internal format */
     width, height, 0,            /* width, height, border
     GL_RGB, GL_UNSIGNED_BYTE,    /* external format, type
     pixels                       /* pixels */
    );

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S,     GL_
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T,     GL_CLAMP_TO_EDGE);

  free(pixels);
  return texture;
}
```

```
// [0] : array buffer, [1] : element buffer
GLuint buffers[NUM_BUFFERS];

GLfloat timer = 0.0;
GLint uniform_timer;

GLuint textures[2];

GLint uniform_textures[2];

GLfloat vertices[][4] = {
  { -1.0, -1.0, 0.0, 1.0},
  {  1.0, -1.0, 0.0, 1.0},
  {  1.0,  1.0, 0.0, 1.0},
  { -1.0,  1.0, 0.0, 1.0}
};

GLubyte indices[4] = {0, 1, 2, 3};
```

< OpenGL ES Application >

Fading a Textured Image (2)

```
void makeTextures()
{
  textures[0] = make_texture("hello1.tga");
  textures[1] = make_texture("hello2.tga");

  uniform_textures[0] =
    glGetUniformLocation(program, "textures[0]");
  uniform_textures[1] =
    glGetUniformLocation(program, "textures[1]");

  uniform_timer = glGetUniformLocation(program, "timer");
}
```

```
void render(ESContext *esContext)
{
  ...

  glUniform1f(uniform_timer*0.0001, timer);

  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, textures[0]);
  glUniform1i(uniform_textures[0], 0);

  glActiveTexture(GL_TEXTURE1);
  glBindTexture(GL_TEXTURE_2D, textures[1]);
  glUniform1i(uniform_textures[1], 1);

  ...

  glDrawElements(GL_POLYGON, 4, GL_UNSIGNED_BYTE,
}
```

< OpenGL ES Application >

```
uniform sampler2D textures[2];

in float fade_factor;
in vec2 texcoord;

out vec4 FragColor;

void main()
{
  FragColor = mix(
      texture(textures[0], texcoord),
      texture(textures[1], texcoord),
      fade_factor
  );
}
```

※ mix(x, y, a) : x*(1.0-a) + y*a

< Fragment Shader >

```
uniform float timer;

layout(location = 0) in vec3 position;

out vec2 texcoord;
out float fade_factor;

void main()
{
  gl_Position = vec4(position, 1.0);
  texcoord = position.xy * vec2(0.5) + vec2(0.5);
  fade_factor = sin(timer) * 0.5 + 0.5;
}
```

< Vertex Shader >

순서

- OpenGL ES Shading Language and Shader
  - EGL & OpenGL ES 3.0 Graphics Pipeline
  - OpenGL ES Shading Language
  - Compiling and Linking Shader Source
  - Vertex Shader
  - Fragment Shader
  - Using Textures in a Shader

- Patterns of Shaders in Android
  - Simple Pattern in Screen Recorder Command
  - Extensible Pattern in Mobile Filter Framework
  - Flexible Pattern in SurfaceFlinger and HWUI

Simple Pattern in Screen Recorder Command

frameworks/av/cmds/screenrecord/Program.cpp

```
// Simple vertex shader.  Texture coord calc includes matrix for
// GLConsumer transform.
static const char* kVertexShader =
        "uniform mat4 uMVPMatrix;\n"
        "uniform mat4 uGLCMatrix;\n"
        "attribute vec4 aPosition;\n"
        "attribute vec4 aTextureCoord;\n"
        "varying vec2 vTextureCoord;\n"
        "void main() {\n"
        "    gl_Position = uMVPMatrix * aPosition;\n"
        "    vTextureCoord = (uGLCMatrix * aTextureCoord).xy;\n"
        "}\n";
```

```
// Trivial fragment shader for mundane texture.
static const char* kFragmentShader =
        "precision mediump float;\n"
        "varying vec2 vTextureCoord;\n"
        "uniform sampler2D uTexture;\n"
        "void main() {\n"
        "    gl_FragColor = texture2D(uTexture, vTextureCoord);\n"
        //"    gl_FragColor = vec4(0.2, 1.0, 0.2, 1.0);\n"
        "}\n";
```

```
// Trivial fragment shader for external texture.
static const char* kExtFragmentShader =
        "#extension GL_OES_EGL_image_external : require\n"
        "precision mediump float;\n"
        "varying vec2 vTextureCoord;\n"
        "uniform samplerExternalOES uTexture;\n"
        "void main() {\n"
        "    gl_FragColor = texture2D(uTexture, vTextureCoord);\n"
        "}\n";
```

```
status_t Program::setup(ProgramType type) {
    ALOGV("Program::setup type=%d", type);
    status_t err;

    mProgramType = type;

    GLuint program;
    if (type == PROGRAM_TEXTURE_2D) {
        err = createProgram(&program, kVertexShader, kFragmentShader);
    } else {
        err = createProgram(&program, kVertexShader, kExtFragmentShader);
    }
    if (err != NO_ERROR) {
        return err;
    }
    assert(program != 0);
    ...
```

Extensible Pattern in Mobile Filter Framework (1)

frameworks/base/media/mca/filterfw/java/android/filterfw/core/ShaderProgram.java

```
public ShaderProgram(FilterContext context, String fragmentShader) {
    mGLEnvironment = getGLEnvironment(context);
    allocate(mGLEnvironment, null, fragmentShader);
    if (!compileAndLink()) {
        throw new RuntimeException("Could not compile and link shader!");
    }
    this.setTimer();
}

public ShaderProgram(FilterContext context, String vertexShader, String fragmentShader) {
    mGLEnvironment = getGLEnvironment(context);
    allocate(mGLEnvironment, vertexShader, fragmentShader);
    if (!compileAndLink()) {
        throw new RuntimeException("Could not compile and link shader!");
    }
    this.setTimer();
}
```

frameworks/base/media/mca/filterfw/jni/jni_shader_program.cpp

```
jboolean Java_android_filterfw_core_ShaderProgram_allocate(JNIEnv* env,
                                                           jobject thiz,
                                                           jobject gl_env,
                                                           jstring vertex_shader,
                                                           jstring fragment_shader) {
  ...
  // Create the shader
  if (!fragment_shader || !gl_env_ptr)
    return false;
  else if (!vertex_shader)
    return ToJBool(WrapObjectInJava(new ShaderProgram(
      gl_env_ptr,
      ToCppString(env, fragment_shader)),
      env, thiz, true));
  else
    return ToJBool(WrapObjectInJava(new ShaderProgram(
      gl_env_ptr,
      ToCppString(env, vertex_shader),
      ToCppString(env, fragment_shader)),
      env, thiz, true));
}
```

frameworks/base/media/mca/filterfw/native/core/shader_program.h

```
class ShaderProgram {
  public:
    explicit ShaderProgram(GLEnv* gl_env,
                           const std::string& fragment_shader);
    ShaderProgram(GLEnv* gl_env,
                  const std::string& vertex_shader,
                  const std::string& fragment_shader);
    bool CompileAndLink();

    // The shader source code
    std::string fragment_shader_source_;
    std::string vertex_shader_source_;
    // The compiled shaders and linked program
    GLuint fragment_shader_;
    GLuint vertex_shader_;
    GLuint program_;
...
```

Extensible Pattern in Mobile Filter Framework (2)

```
public class AlphaBlendFilter extends ImageCombineFilter {
    private final String mAlphaBlendShader =
            "precision mediump float;\n" +
            "uniform sampler2D tex_sampler_0;\n" +
            "uniform sampler2D tex_sampler_1;\n" +
            "uniform sampler2D tex_sampler_2;\n" +
            "uniform float weight;\n" +
            "varying vec2 v_texcoord;\n" +
            "void main() {\n" +
            "  vec4 colorL = texture2D(tex_sampler_0, v_texcoord);\n" +
            "  vec4 colorR = texture2D(tex_sampler_1, v_texcoord);\n" +
            "  float blend = texture2D(tex_sampler_2, v_texcoord).r * weight;\n" +
            "  gl_FragColor = colorL * (1.0 - blend) + colorR * blend;\n" +
            "}\n";

    public AlphaBlendFilter(String name) {
        super(name, new String[] { "source", "overlay", "mask" }, "blended", "weight");
    }

    @Override protected Program getNativeProgram(FilterContext context) {
        throw new RuntimeException("TODO: Write native implementation for AlphaBlend!");
    }

    @Override protected Program getShaderProgram(FilterContext context) {
        return new ShaderProgram(context, mAlphaBlendShader);
    }

}
```

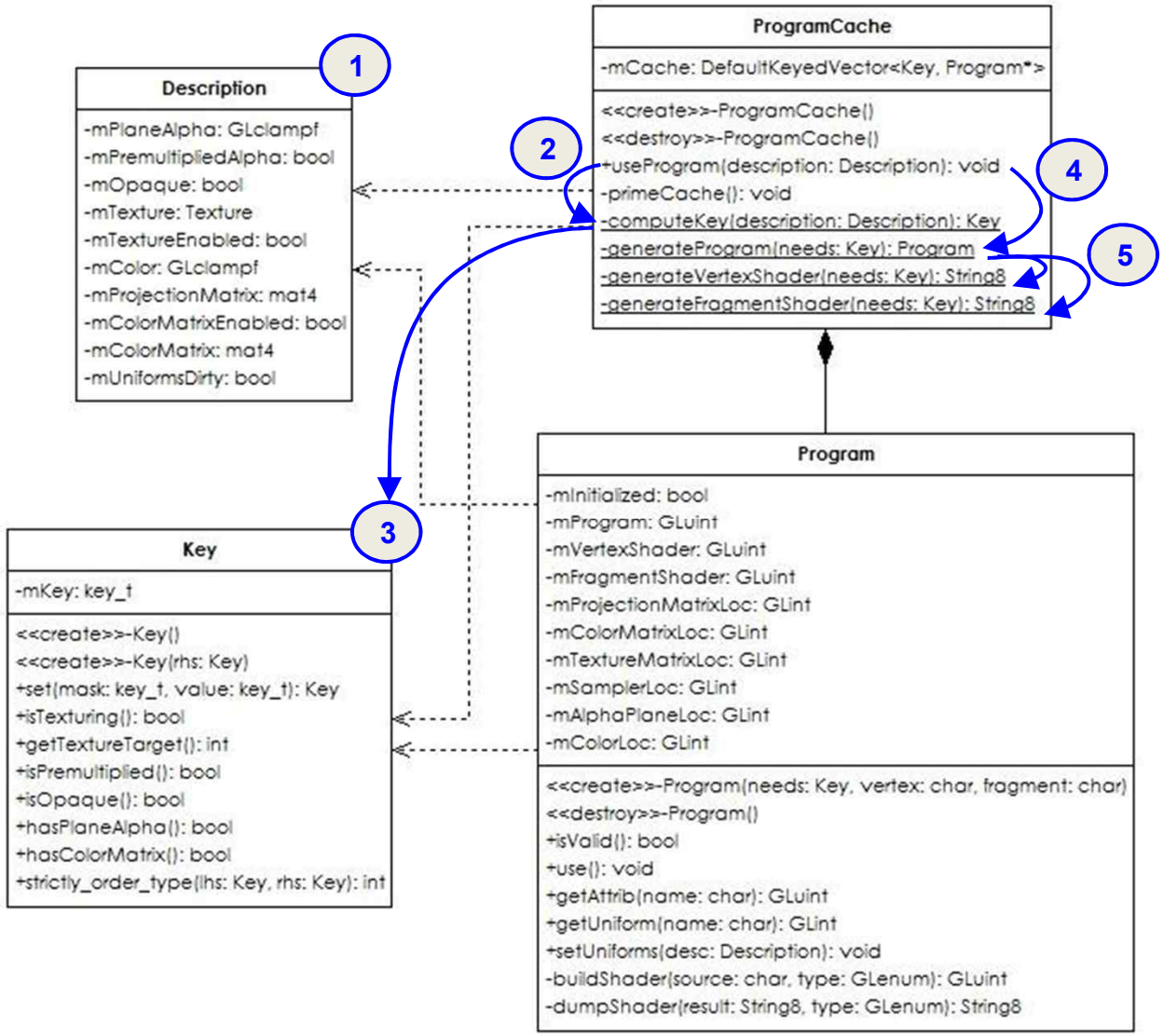
android.filterpacks.imageproc.AlphaBlendFilter
android.filterpacks.imageproc.AutoFixFilter
android.filterpacks.imageproc.BitmapOverlayFilter
android.filterpacks.imageproc.BlackWhiteFilter
android.filterpacks.imageproc.BlendFilter
android.filterpacks.imageproc.BrightnessFilter
android.filterpacks.imageproc.ColorTemperatureFilter
android.filterpacks.imageproc.ContrastFilter
android.filterpacks.imageproc.CropFilter
android.filterpacks.imageproc.CropRectFilter
android.filterpacks.imageproc.CrossProcessFilter
android.filterpacks.imageproc.DocumentaryFilter
android.filterpacks.imageproc.DrawOverlayFilter
android.filterpacks.imageproc.DrawRectFilter
android.filterpacks.imageproc.DuotoneFilter

android.filterpacks.imageproc.FillLightFilter
android.filterpacks.imageproc.FisheyeFilter
android.filterpacks.imageproc.FixedRotationFilter
android.filterpacks.imageproc.FlipFilter
android.filterpacks.imageproc.GrainFilter
android.filterpacks.imageproc.ImageSlicer
android.filterpacks.imageproc.ImageStitcher
android.filterpacks.imageproc.Invert
android.filterpacks.imageproc.LomoishFilter
android.filterpacks.imageproc.NegativeFilter
android.filterpacks.imageproc.PosterizeFilter
android.filterpacks.imageproc.RedEyeFilter
android.filterpacks.imageproc.ResizeFilter
android.filterpacks.imageproc.RotateFilter
android.filterpacks.imageproc.SaturateFilter

android.filterpacks.imageproc.SepiaFilter
android.filterpacks.imageproc.SharpenFilter
android.filterpacks.imageproc.StraightenFilter
android.filterpacks.imageproc.TintFilter
android.filterpacks.imageproc.ToGrayFilter
android.filterpacks.imageproc.ToPackedGrayFilter
android.filterpacks.imageproc.VignetteFilter
android.filterpacks.ui.SurfaceRenderFilter
android.filterpacks.ui.SurfaceTargetFilter
android.filterpacks.videoproc.BackDropperFilter
android.filterpacks.videosink.MediaEncoderFilter
android.filterpacks.videosrc.CameraSource
android.filterpacks.videosrc.MediaSource
android.filterpacks.videosrc.SurfaceTextureSource
android.filterpacks.videosrc.SurfaceTextureTarget

Flexible Pattern in SurfaceFlinger (1)

```
void GLES20RenderEngine::drawMesh(const Mesh& mesh) {
    ProgramCache::getInstance().useProgram(mState);
    …
}
```

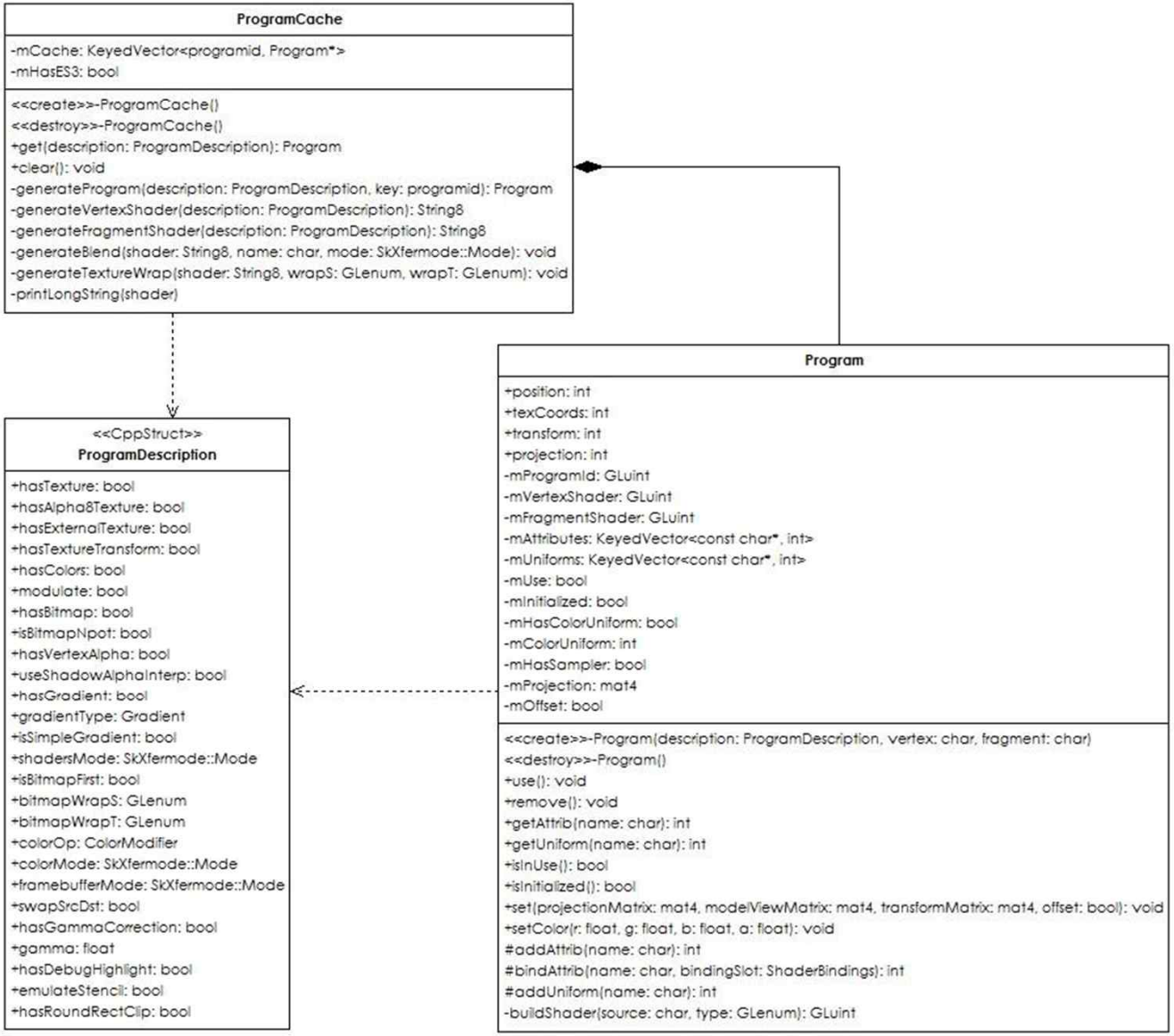


Flexible Pattern in SurfaceFlinger (2)

```
String8 ProgramCache::generateVertexShader(const Key& needs) {
    Formatter vs;
    if (needs.isTexturing()) {
        vs  << "attribute vec4 texCoords;"
            << "varying vec2 outTexCoords;";
    }
    vs << "attribute vec4 position;"
       << "uniform mat4 projection;"
       << "uniform mat4 texture;"
       << "void main(void) {" << indent
       << "gl_Position = projection * position;";
    if (needs.isTexturing()) {
        vs << "outTexCoords = (texture * texCoords).st;";
    }
    vs << dedent << "}";
    return vs.getString();
}
```

```
String8 ProgramCache::generateFragmentShader(const Key& needs) {
    Formatter fs;
    if (needs.getTextureTarget() == Key::TEXTURE_EXT) {
        fs << "#extension GL_OES_EGL_image_external : require";
    }

    // default precision is required-ish in fragment shaders
    fs << "precision mediump float;";

    if (needs.getTextureTarget() == Key::TEXTURE_EXT) {
        fs << "uniform samplerExternalOES sampler;"
           << "varying vec2 outTexCoords;";
    } else if (needs.getTextureTarget() == Key::TEXTURE_2D) {
        fs << "uniform sampler2D sampler;"
           << "varying vec2 outTexCoords;";
    } else if (needs.getTextureTarget() == Key::TEXTURE_OFF) {
        fs << "uniform vec4 color;";
    }
    if (needs.hasPlaneAlpha()) {
        fs << "uniform float alphaPlane;";
    }
    if (needs.hasColorMatrix()) {
        fs << "uniform mat4 colorMatrix;";
    }
    fs << "void main(void) {" << indent;
    if (needs.isTexturing()) {

    …

    fs << dedent << "}";
    return fs.getString();
}
```

Flexible Pattern in HWUI (1)



Flexible Pattern in HWUI (2)

```
String8 ProgramCache::generateVertexShader(const ProgramDescription& description) {
    // Add attributes
    String8 shader(gVS_Header_Attributes);
    if (description.hasTexture || description.hasExternalTexture) {
        shader.append(gVS_Header_Attributes_TexCoords);
    }
    if (description.hasVertexAlpha) {
        shader.append(gVS_Header_Attributes_VertexAlphaParameters);
    }
    if (description.hasColors) {
        shader.append(gVS_Header_Attributes_Colors);
    }
    ...
    shader.append(gVS_Footer);

    PROGRAM_LOGD("*** Generated vertex shader:\n\n%s", shader.string());

    return shader;
}
```

```
// Vertex shaders snippets
const char* gVS_Header_Attributes =
        "attribute vec4 position;\n";
const char* gVS_Header_Attributes_TexCoords =
        "attribute vec2 texCoords;\n";
const char* gVS_Header_Attributes_Colors =
        "attribute vec4 colors;\n";
const char* gVS_Header_Attributes_VertexAlphaParameters =
        "attribute float vtxAlpha;\n";
const char* gVS_Header_Uniforms_TextureTransform =
        "uniform mat4 mainTextureTransform;\n";
...
// Fragment shaders snippets
const char* gFS_Header_Extension_FramebufferFetch =
        "#extension GL_NV_shader_framebuffer_fetch : enable\n\n";
const char* gFS_Header_Extension_ExternalTexture =
        "#extension GL_OES_EGL_image_external : require\n\n";
const char* gFS_Header =
        "precision mediump float;\n\n";
const char* gFS_Uniforms_Color =
        "uniform vec4 color;\n";
...
```

```
String8 ProgramCache::generateFragmentShader(const ProgramDescription& description) {
    String8 shader;

    const bool blendFramebuffer = description.framebufferMode >= SkXfermode::kPlus_Mode;
    if (blendFramebuffer) {
        shader.append(gFS_Header_Extension_FramebufferFetch);
    }
    if (description.hasExternalTexture) {
        shader.append(gFS_Header_Extension_ExternalTexture);
    }
    ...
    shader.append(gFS_Footer);

#if DEBUG_PROGRAMS
        PROGRAM_LOGD("*** Generated fragment shader:\n\n");
        printLongString(shader);
#endif

    return shader;
}
```

Flexible Pattern in HWUI (3)

```
status_t OpenGLRenderer::drawVertexBuffer(float translateX, float translateY,
        const VertexBuffer& vertexBuffer, const SkPaint* paint, int displayFlags) {
    ...
    setupDraw();
    setupDrawNoTexture();
    if (isAA) setupDrawVertexAlpha((displayFlags & kVertexBuffer_ShadowInterp));
    setupDrawColor(color, ((color >> 24) & 0xFF) * mSnapshot->alpha);
    setupDrawColorFilter(getColorFilter(paint));
    setupDrawShader(getShader(paint));
    setupDrawBlending(paint, isAA);
    setupDrawProgram();
    setupDrawModelView(kModelViewMode_Translate, (displayFlags & kVertexBuffer_Offset),
            translateX, translateY, 0, 0);
    setupDrawColorUniforms(getShader(paint));
    setupDrawColorFilterUniforms(getColorFilter(paint));
    setupDrawShaderUniforms(getShader(paint));
    ...
```

```
void OpenGLRenderer::setupDrawColorFilter(const SkColorFilter* filter) {
    ...
    SkXfermode::Mode mode;
    if (filter->asColorMode(NULL, &mode)) {
        mDescription.colorOp = ProgramDescription::kColorBlend;
        mDescription.colorMode = mode;
    } else if (filter->asColorMatrix(NULL)) {
        mDescription.colorOp = ProgramDescription::kColorMatrix;
    }
}
```

```
void OpenGLRenderer::setupDrawColorFilterUniforms(const SkColorFilter* filter) {
    SkColor color;
    SkXfermode::Mode mode;
    if (filter->asColorMode(&color, &mode)) {
        glUniform4f(mCaches.currentProgram->getUniform("colorBlend"), r, g, b, a);
        return;
    }
    SkScalar srcColorMatrix[20];
    if (filter->asColorMatrix(srcColorMatrix)) {
        glUniformMatrix4fv(mCaches.currentProgram->getUniform("colorMatrix"), 1,
                GL_FALSE, colorMatrix);
        glUniform4fv(mCaches.currentProgram->getUniform("colorMatrixVector"), 1, colorVector);
        return;
    }
}
```

References

1. OpenGL Programming Guide (7e) // OpenGL version 3.0 and 3.1
2. OpenGL Shading Language (3e) // GLSL 1.4
3. OpenGL ES 3.0 Programming Guide (2e)
4. http://duriansoftware.com/joe/An-intro-to-modern-OpenGL.-Table-of-Contents.html
5. http://ogldev.atspace.co.uk/index.html