Post-Processing with OpenGL ES 2.0

Hua Thi Le - le.huathi@gameloft.com

Jul 9th 2012

GAMELOFT

Contents

* Post-processing concepts
* Framebuffer Object
* Common post-processing techniques

GAMELOFT

Contents

* **Post-processing concepts**
* Framebuffer Object
* Common post-processing techniques

GAMELOFT

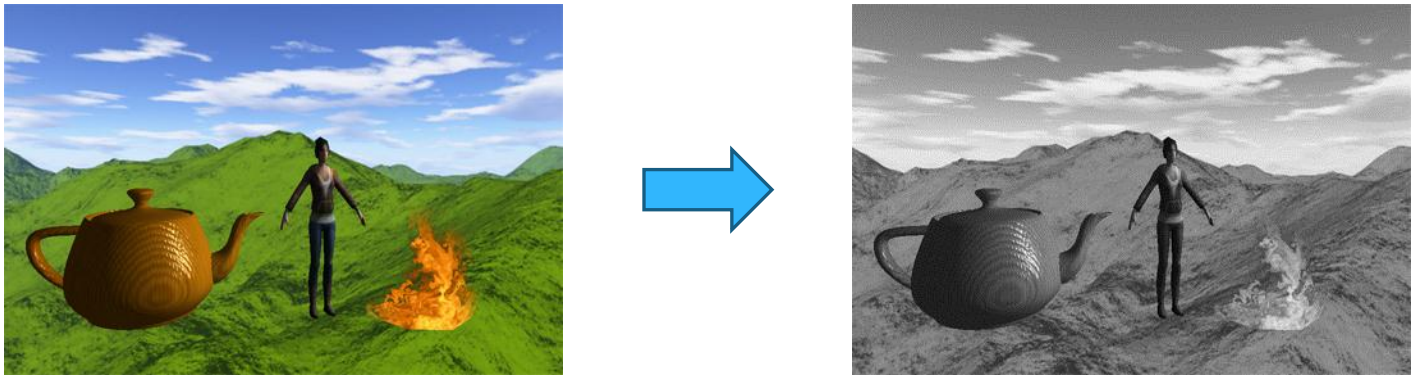
Post-processing concepts

* Why use post-processing?
* How to do it?

GAMELOFT

Why use post-processing ?

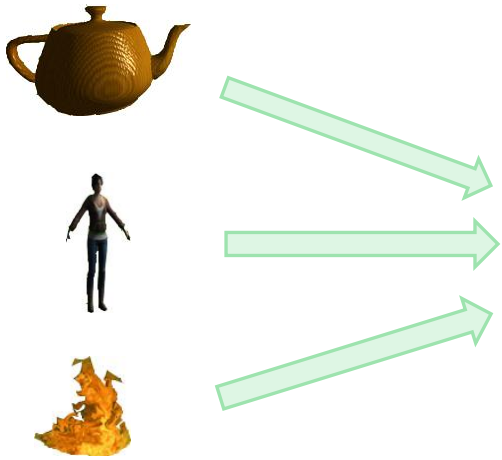
* **Purpose:** apply a **global effect** on the **whole scene**



GAMELOFT

How to do post-processing

* Normal rendering:



Screen

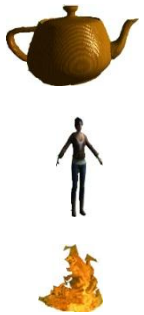
Pipelines for each object

What's pipeline?

GAMELOFT

How to do post-processing

* **Render to textures -> process -> render to screen**

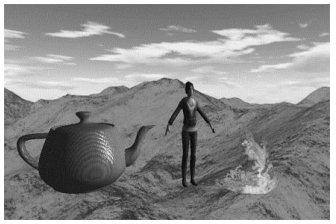


Pipelines for each object

Rendered textures

Pipeline for whole screen

Screen / textures



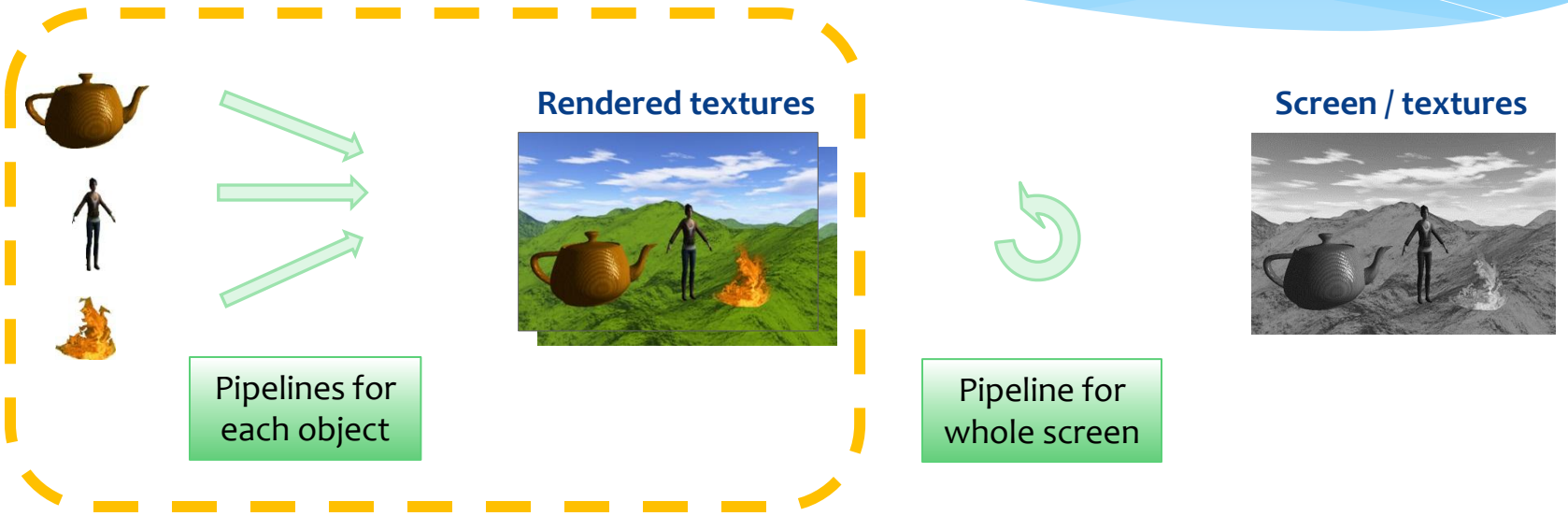
How to render this?

How to process it?

GAMELOFT

How to do post-processing

* **Step 1:** Rendering to textures

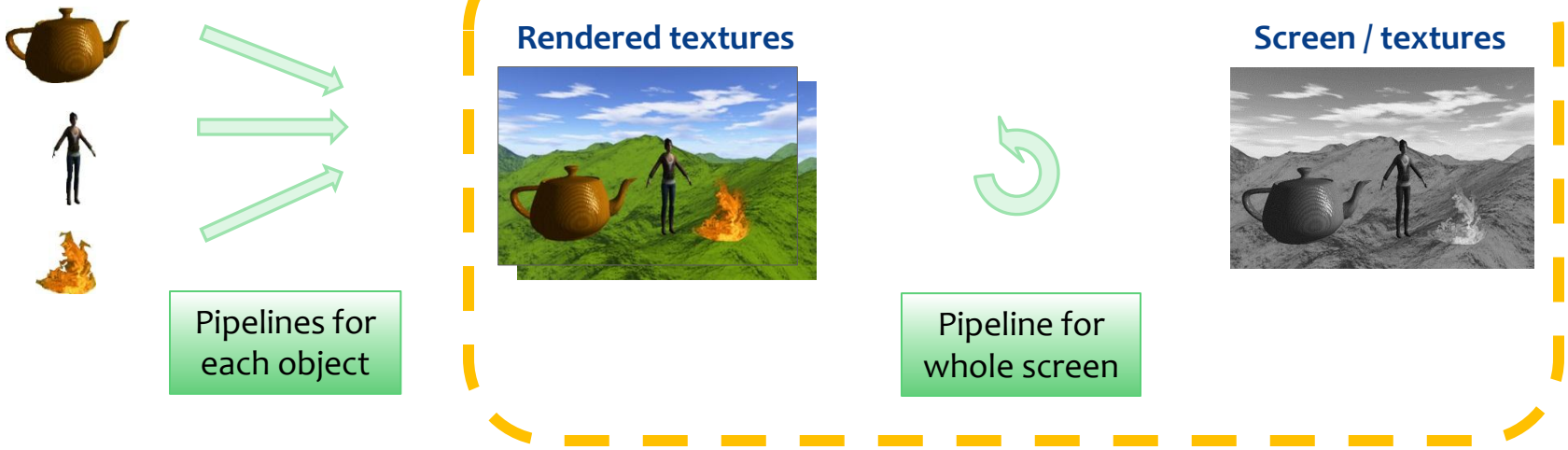


  * Use an off-screen **Frame Buffer Object**
  * Result: **textures** (color, depth, scissor)

GAMELOFT

How to do post-processing

* **Step 2:** Draw a quad with specific processing



* **Input:** textures from step 1
* **Processing:** in fragment shader
* **Output:** other textures or screen

GAMELOFT

How to do post-processing

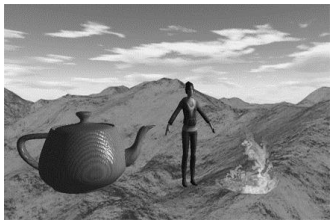
* **Step 2** (continue)

Pipelines for each object

Rendered textures

Pipeline for whole screen

Screen / textures



* Step 2 can **loop** a few times
* The last step **must** draw to screen

GAMELOFT

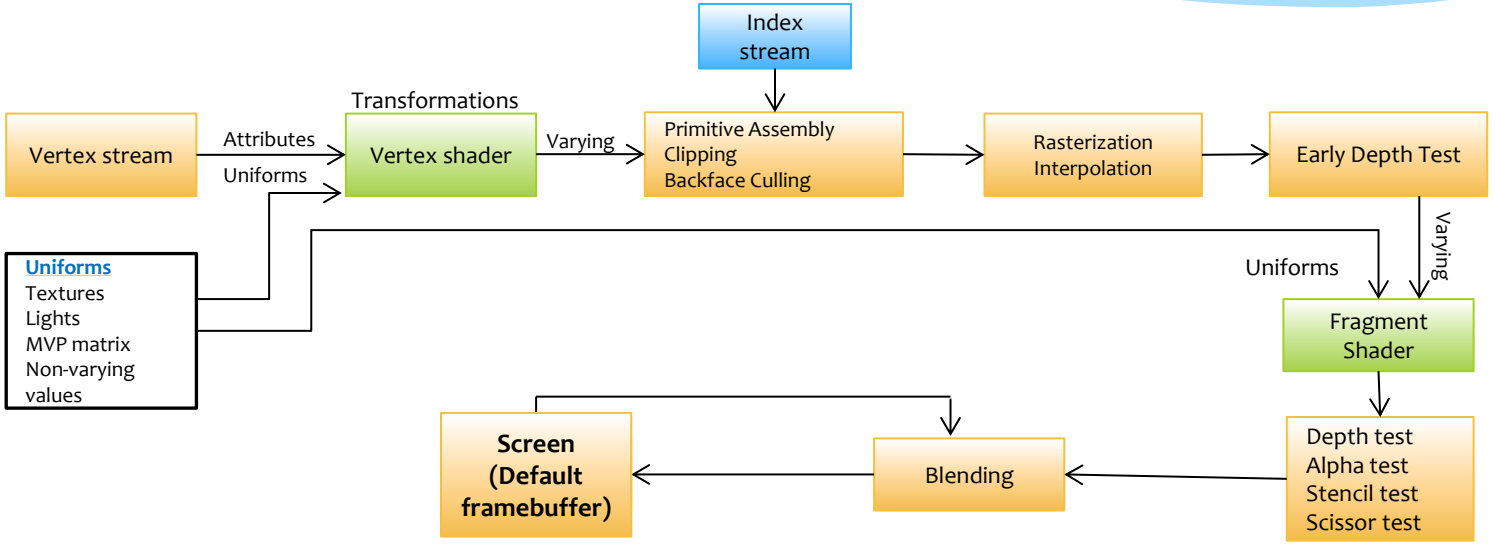
Contents

* Post-processing concepts
* **Framebuffer Object**
* Common post-processing techniques

GAMELOFT

Framebuffer Object

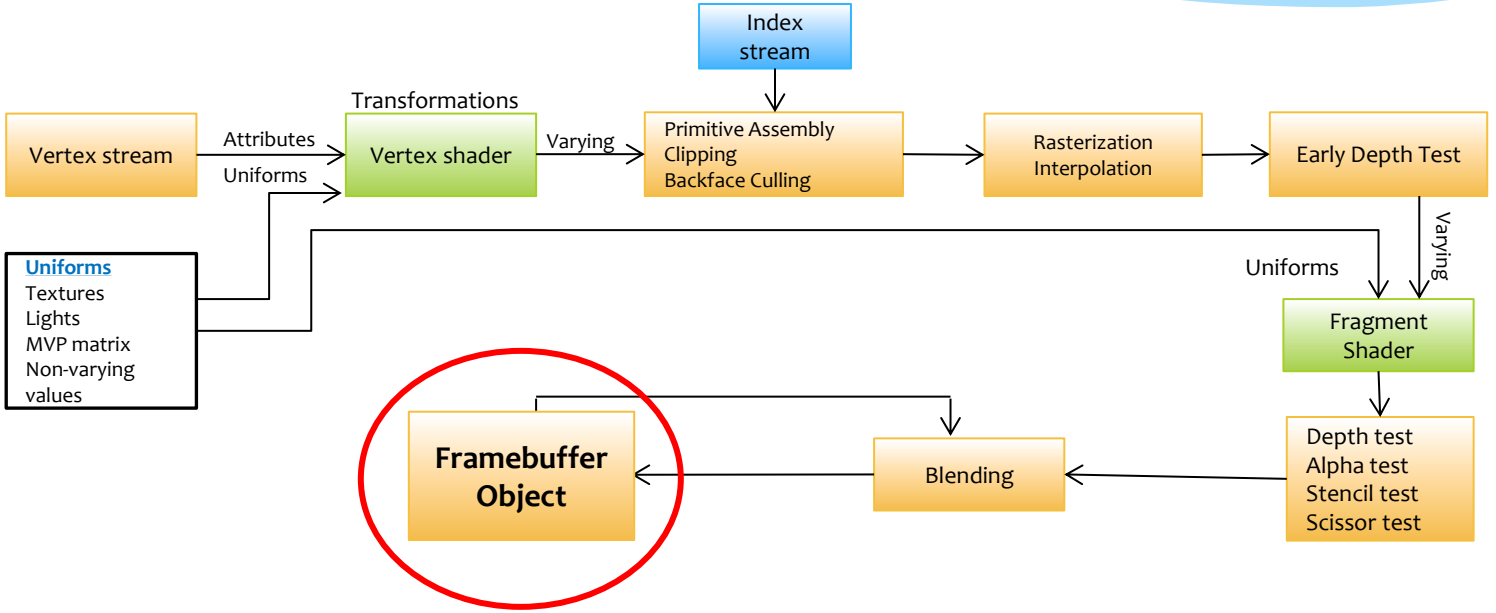
* Render to default framebuffer:



GAMELOFT

Framebuffer Object

* Render to FBO :

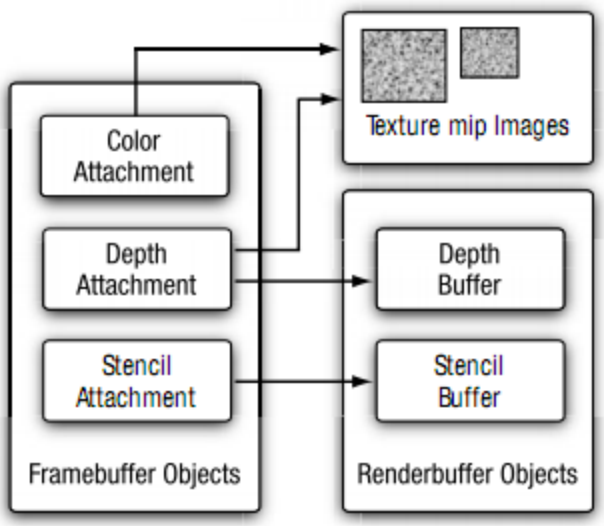


Replace the back-buffer render target with an **off-screen target**

GAMELOFT

Framebuffer Object

* FBO structure:



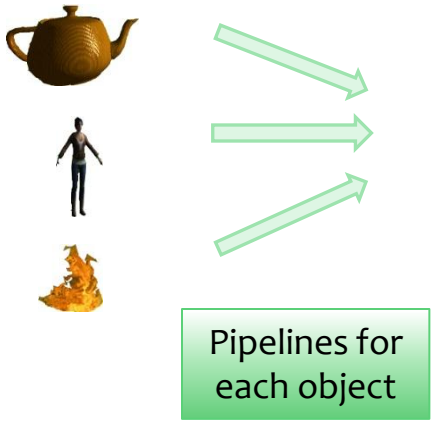
* **Attachment:** attach (register) to get **rendering result**
* **3 attachment types:**
  * Color
  * Depth
  * Stencil
* **2 object types:**
  * Texture Object
  * Renderbuffer Object

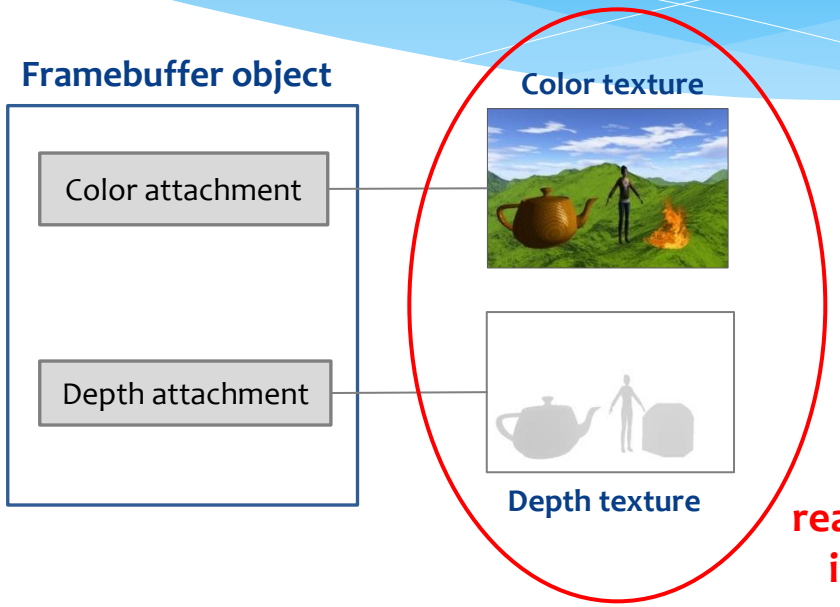
- Can have more than 1 color attachment
- Depth attachment can be texture object or renderbuffer object

GAMELOFT

Framebuffer Object

* Render to FBO:



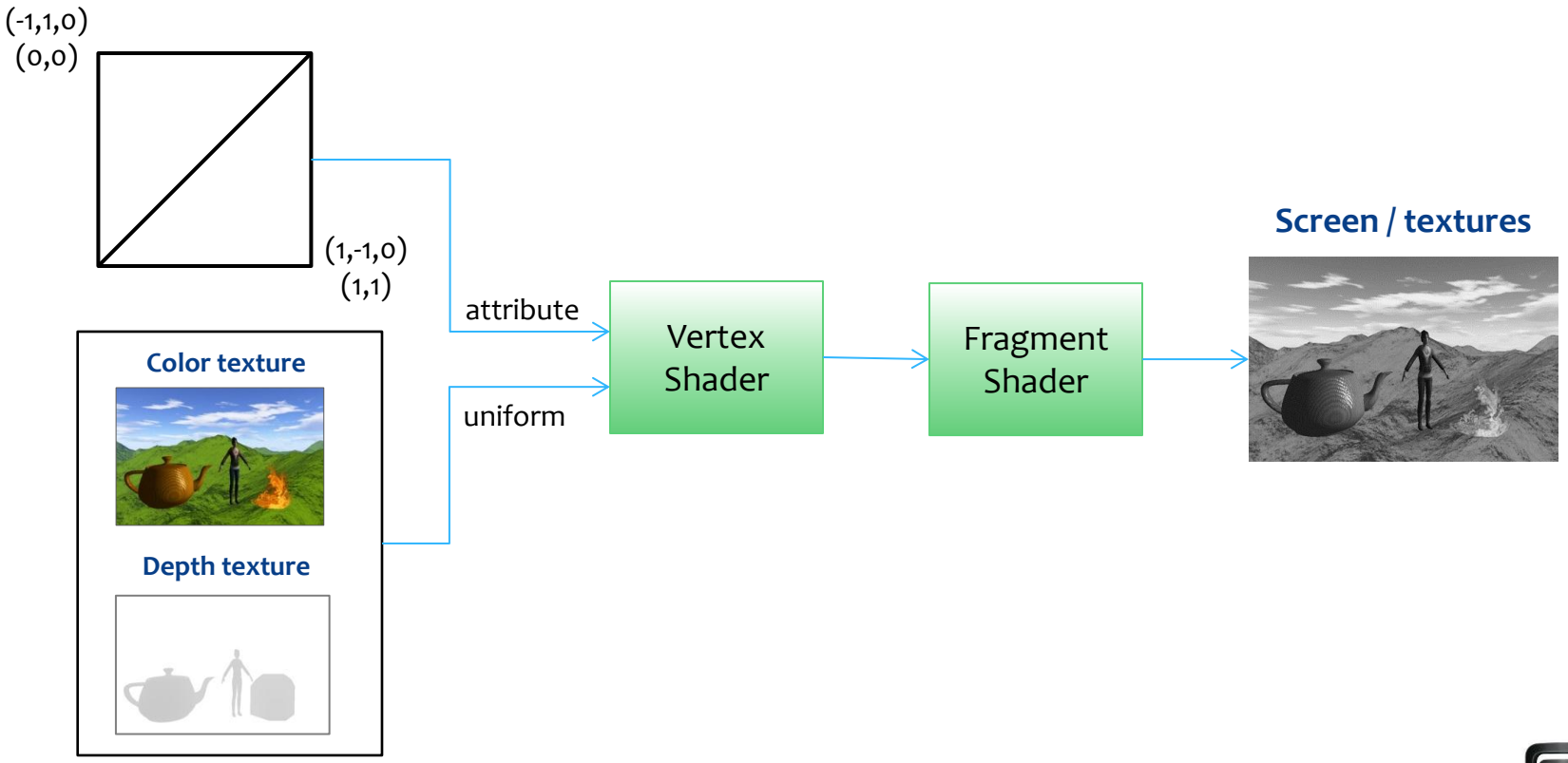


Note: MUST initialize FBO and textures **outside** the render function

GAMELOFT

Framebuffer Object

* Step 2: Draw textures to screen / other textures:



GAMELOFT

Framebuffer Object (coding)

* Create FBO
* Set current FBO to use
* Return to system default framebuffer
* Attach objects to FBO (textures, renderbuffers)
* Render objects to FBO
* Delete FBO

GAMELOFT

Framebuffer Object (coding)

* Create FBO:

```
GLuint fboId;
glGenFramebuffers(1, &fboId);
```

* Set current FBO to use:

```
glBindFramebuffer(GL_FRAMEBUFFER, fboId);
```

* Return to system default framebuffer (screen):

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

GAMELOFT

Framebuffer Object (coding)

* Attach color texture to FBO:

```
//generate color texture
glGenTextures(1, &colorTexId);
glBindTexture(GL_TEXTURE_2D, colorTexId);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

//initialize an empty texture with screen width & height
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA,
GL_UNSIGNED_BYTE, NULL);

//attach texture to GL_COLOR_ATTACHMENT0
glBindFramebuffer(GL_FRAMEBUFFER, fboId);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D,
colorTexId, 0);

//bind system default texture
glBindTexture(GL_TEXTURE_2D, 0);
```

GAMELOFT

Framebuffer Object (coding)

* Attach depth texture to FBO:

```
//generate depth texture - same as color texture
glGenTextures(1, &depthTexId);
glBindTexture(GL_TEXTURE_2D, depthTexId);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);

//initialize an empty texture with screen width & height
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, width, height, 0,
GL_DEPTH_COMPONENT, GL_UNSIGNED_INT, NULL);

//attach texture to GL_DEPTH_ATTACHMENT
glBindFramebuffer(GL_FRAMEBUFFER, fboId);
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,
depthTexId, 0);

//bind system default texture
glBindTexture(GL_TEXTURE_2D, 0);
```

GAMELOFT

Framebuffer Object (coding)

* Render objects to FBO:

```
//bind the target FBO
glBindFramebuffer(GL_FRAMEBUFFER, fboId);

//render each objects in list object
for(vector<Object*>::iterator it=m_vObjects.begin();it!=m_vObjects.end();++it)
{
    (*it)->Render(globalTime, &m_maProjection, pmaView, &m_camera);
}

//draw post-effect
```

* Delete FBO:

```
glDeleteFramebuffers(1, &fboId);
```

GAMELOFT

Contents

* Post-processing concepts
* Framebuffer Object
* **Common post-processing techniques**

GAMELOFT

Common techniques

* Black-white
* Blur
* Bloom
* Depth of field (DOF)
* Edge detection
* Shadow
* Complex reflection
* Snow

GAMELOFT

Black - white

* Run step 2 once
* Simple grayscale in fragment shader

```
vec4 color = texture2D(u_texture, v_uv);
float t = 0.3 * color.r + 0.59 * color.g + 0.11 * color.b;
//float t = 0.33 * color.r + 0.33 * color.g + 0.34 * color.b;

gl_FragColor = vec4(t, t, t, 1.0);
```

GAMELOFT

Blur

* Apply a blur filter for each pixel

```
uniform colorTexture;
uniform step;
varying v_uv;

void main(void)
{
   color1 = sampling2Dfrom(colorTexture, v_uv);
   color2 = sampling2Dfrom(colorTexture, (v_uv.x + step.x, v_uv.y));
   color3 = sampling2Dfrom(colorTexture, (v_uv.x - step.x, v_uv.y));
   color4 = sampling2Dfrom(colorTexture, (v_uv.x, v_uv.y + step.y));
   color5 = sampling2Dfrom(colorTexture, (v_uv.x, v_uv.y - step.y));
   color6 = sampling2Dfrom(colorTexture, (v_uv.x + step.z, v_uv.y + step.w));
   color7 = sampling2Dfrom(colorTexture, (v_uv.x - step.z, v_uv.y + step.w));
   color8 = sampling2Dfrom(colorTexture, (v_uv.x - step.z, v_uv.y - step.w));
   color9 = sampling2Dfrom(colorTexture, (v_uv.x + step.z, v_uv.y - step.w));
   gl_FragColor = (color1 * 2.0 + color2 + color3 + color4 + color5 + color6 + color7
+ color8 + color9) * 0.1;
}
```

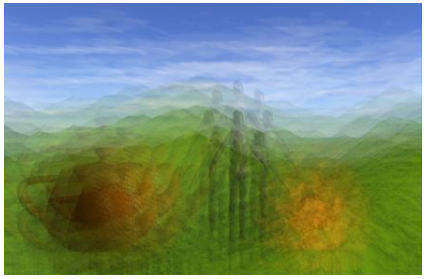
GAMELOFT

Blur (cont.)

* Pass step to shader by a multiplier **k**

```
if((i32Location = glGetUniformLocation(program, "step")) != -1)
{
   float x = 1.0f / pESContext->width;
   float y = 1.0f / pESContext->height;
   float z = sqrt(2.0f) / 2.0f * x;
   float w = sqrt(2.0f) / 2.0f * y;
   glUniform4f(i32Location, k * x, k * y, k * z, k * w);
}
```

k = 1.0

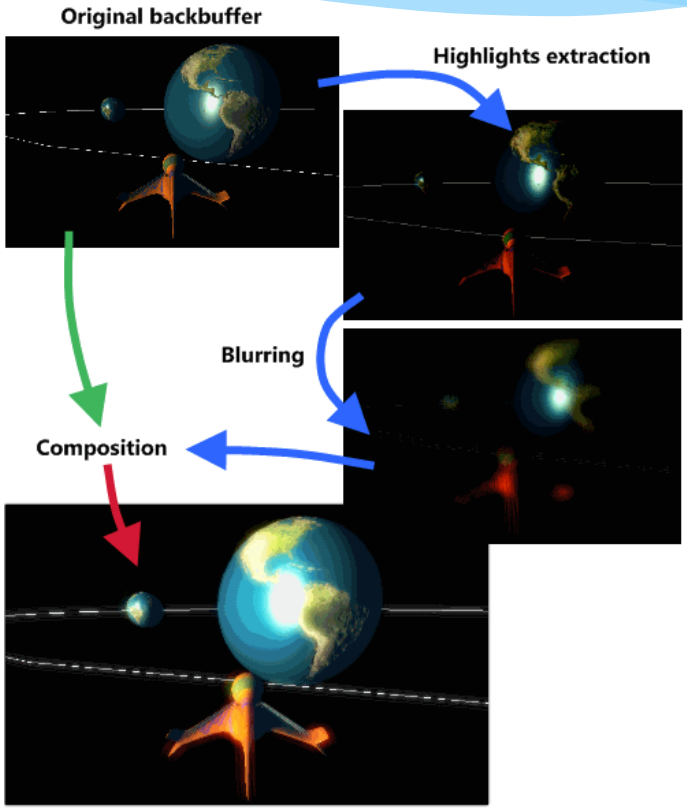


k = 63.0

GAMELOFT

Bloom

* Bloom: overexpose effect



GAMELOFT

Bloom

* Implement: 3 steps
  * **Pre-bloom:** extract the bright part
  * **Blur:** blur this bright part several times (common is 5)
  * **Post-bloom:** blend initial color buffer with blur result

Sample code:

```
g_postEffPreBloom.DrawPostEffect(esContext, &fbo2, fbo1.fbo_color, fbo1.fbo_depth, 0, 0);

g_postEffBlur.DrawPostEffect( esContext, &fbo3, fbo2.fbo_color, fbo2.fbo_depth, 0, 5 );
g_postEffBlur.DrawPostEffect( esContext, &fbo2, fbo3.fbo_color, fbo3.fbo_depth, 0, 17 );
g_postEffBlur.DrawPostEffect( esContext, &fbo3, fbo2.fbo_color, fbo2.fbo_depth, 0, 31 );
g_postEffBlur.DrawPostEffect( esContext, &fbo2, fbo3.fbo_color, fbo3.fbo_depth, 0, 43 );
g_postEffBlur.DrawPostEffect( esContext, &fbo3, fbo2.fbo_color, fbo2.fbo_depth, 0, 63 );

g_postEffPostBloom.DrawPostEffect( esContext, NULL, fbo1.fbo_color, fbo3.fbo_color, 0, 0 );
```

* Need **more FBO** to do bloom

GAMELOFT

Bloom

* Sample code for pre-bloom: highlights extraction

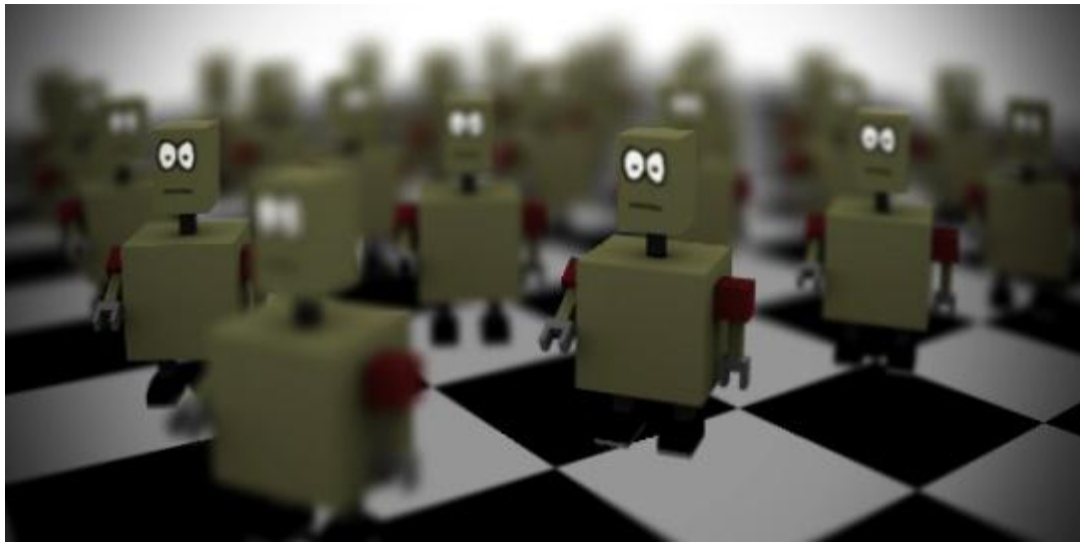
```
uniform sampler2D colorTex;
uniform float limit;
varying mediump vec2 v_texCoords;

void main(void)
{
          vec3 color = sampling2Dfrom(colorTexture, v_texCoords).rgb;
          float brightness = 0.3 * color.r + 0.59 * color.g + 0.11 * color.b;
          float val = step(limit, brightness); //Extract bright fragment

          gl_FragColor = vec4(color * val, 1.0);
}
```

* Post-bloom: mix the original texture with blurred texture

GAMELOFT

Depth of field

* Blur things that **out-of-focus**



GAMELOFT

Depth of field

* Implement: 2 steps
  * Blur: several times (common is 3)
  * DOF: combine original and blur with depth buffer

Sample code:

```
g_postEffBlur.DrawPostEffect(esContext, &fbo2, fbo1.fbo_color, fbo1.fbo_depth, 0, 5);
g_postEffBlur.DrawPostEffect(esContext, &fbo3, fbo2.fbo_color, fbo1.fbo_depth, 0, 9);
g_postEffBlur.DrawPostEffect(esContext, &fbo2, fbo3.fbo_color, fbo1.fbo_depth, 0, 16);

g_postEffDOF.DrawPostEffect(esContext, NULL, fbo1.fbo_color, fbo1.fbo_depth,
fbo2.fbo_color, 0);
```

GAMELOFT

Depth of field

* For better result, use different blur shader algorithm

Sample code:

```
//Get uv coordinate of related fragment
uv2 = v_uv + vec2(u_blurStep.x,0.0);
uv3 = v_uv + vec2(u_blurStep.z, u_blurStep.w);
uv4 = v_uv + vec2(0.0, u_blurStep.y);
uv5 = v_uv + vec2(-u_blurStep.z, u_blurStep.w);
uv6 = v_uv + vec2(-u_blurStep.x, 0.0);
uv7 = v_uv + vec2(-u_blurStep.z, -u_blurStep.w);
uv8 = v_uv + vec2(0.0, -u_blurStep.y);
uv9 = v_uv + vec2(u_blurStep.z, -u_blurStep.w);

//Blur the blurred texture
color1 = sampling2Dfrom(colorTexture, v_uv);
color2 = sampling2Dfrom(colorTexture, uv2);
color3 = sampling2Dfrom(colorTexture, uv3);
color4 = sampling2Dfrom(colorTexture, uv4);
color5 = sampling2Dfrom(colorTexture, uv5);
color6 = sampling2Dfrom(colorTexture, uv6);
color7 = sampling2Dfrom(colorTexture, uv7);
color8 = sampling2Dfrom(colorTexture, uv8);
color9 = sampling2Dfrom(colorTexture, uv9);
```

Depth of field

* For better result, use different blur shader algorithm

Sample code:

```
//Calculate mix factor for each fragment
float d1 = CalculateMixFactor(v_uv);
float d2 = CalculateMixFactor(uv2);
float d3 = CalculateMixFactor(uv3);
float d4 = CalculateMixFactor(uv4);
float d5 = CalculateMixFactor(uv5);
float d6 = CalculateMixFactor(uv6);
float d7 = CalculateMixFactor(uv7);
float d8 = CalculateMixFactor(uv8);
float d9 = CalculateMixFactor(uv9);

float total = 2.0  + d2 + d3 + d4 + d5 + d6 + d7 + d8 + d9;
gl_FragColor = (2.0  * color1 + color2 * d2 + color3 * d3 + color4 * d4 + color5 * d5 +
color6 * d6 + color7 * d7 + color8 * d8 + color9 * d9) / total;
```

GAMELOFT

Depth of field

* Sample code to calculate mix factor

Sample code:

```
Uniform u_dof_near ; //same as near plan of projection - need to be put in uniform
Uniform u_dof_far ;   //same as far plan of projection - need to be put in uniform
Uniform u_dof_clarity; //Set clarity point from C++ code

float CalculateMixFactor(vec2 uv)
{
//Get fragment's depth value from depth buffer
float depth = sampling2Dfrom(depthTexture, uv).x;
//Calculate z
float z = -u_dof_far * u_dof_near / (depth * (u_dof_far - u_dof_near) - u_dof_far);
//Calculate mix factor between z and clarity point
float factor = clamp(abs(z - u_dof_clarity) / u_dof_fade, 0.0, 1.0);
return factor;

}
```

GAMELOFT

Depth of field

* DoF shader is similar to blur shader but need one more step to mix blur textured with original scene

Sample code:

```
vec4 colorOriginal = sampling2Dfrom(originalSceneTexture, v_uv);

vec4 colorBlur = (2.0 * color1 + color2 * d2 + color3 * d3 + color4 * d4 + color5 * d5 +
color6 * d6 + color7 * d7 + color8 * d8 + color9 * d9) / total;

gl_FragColor = mix(colorOriginal,colorBlur, d1);
```

GAMELOFT

Questions & Answers

GAMELOFT