Render to texture

Why?

- Model a camera within a scene
- Compute environment maps on the fly

Framebuffer objects

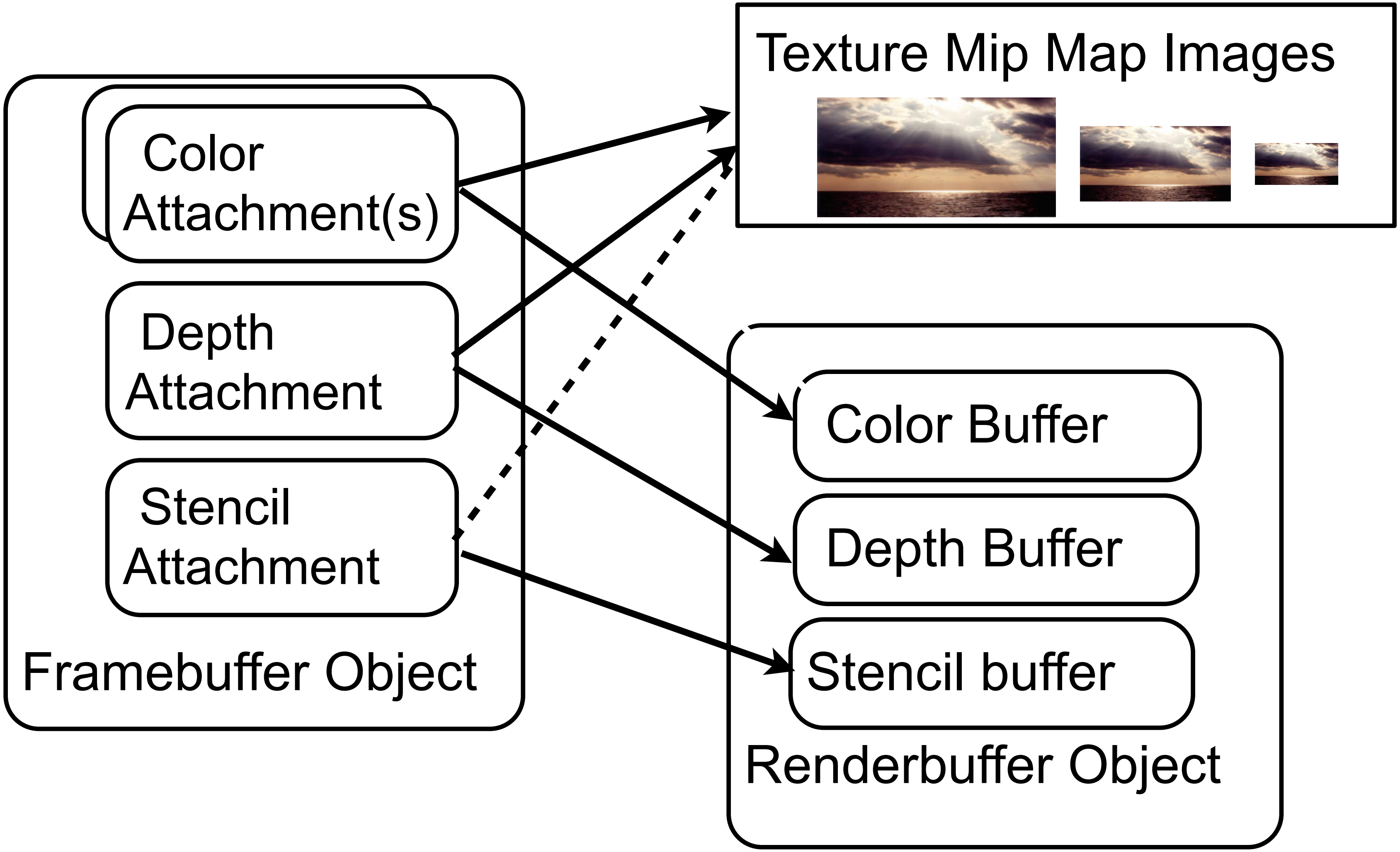
Dynamically allocated frame buffers

- Attachments: Color (many) , Depth (one), Stencil (one)

RenderBuffers or Textures

- The actual storage space. Can be textures or Renderbuffers

Framebuffer attachments



Textures vs Renderbuffers

- Using directly textures as attachments requires simpler code
- Using Renderbuffers is slightly more complex but more elegant and powerful

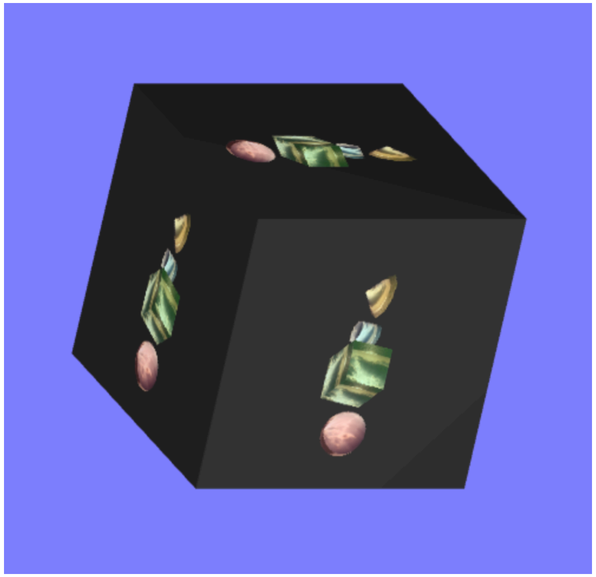
- We will discuss the case with textures

Framebuffer Object vs Window System Framebuffer

- FBOs Pixel ownership test always succeeds
- FBOs can share depth and stencil buffers - new versions of OpenGL support double buffer attachments
- FBOs support multisample buffers as attachments

Two Pass Process

- Pass 1: Bind to the FBO
  Render the texture
- Pass 2: Unbind the FBO (back to default frame buffer)
  Render the scene using the texture from pass 1



Set up the FBO

```
// Create and bind the framebuffer
fbo = gl.createFramebuffer();
gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);

// create the texture and bind it
targetTextureWidth = 256;
targetTextureHeight = 256;
targetTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, targetTexture);

{
    // set up and the texture and attach it
    … next slide
}
```

Texture and attachment

```
{
      // define size and format of level 0
      const level = 0;
      const internalFormat = gl.RGBA;
      const border = 0;
      const format = gl.RGBA;
      const type = gl.UNSIGNED_BYTE;
      const data = null;
      gl.texImage2D(gl.TEXTURE_2D, level, internalFormat,
            targetTextureWidth, targetTextureHeight, border,format, type, data);

      // set the filtering so we don't need mips
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

      // attach the texture as the first color attachment
      const attachmentPoint = gl.COLOR_ATTACHMENT0;
      gl.framebufferTexture2D(
          gl.FRAMEBUFFER, attachmentPoint, gl.TEXTURE_2D, targetTexture, level);
}
```

Depth texture

```
// create a depth texture
depthTexture = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, depthTexture);
// make a depth buffer and the same size as the targetTexture
{
      // define size and format of level 0
      const level = 0;
      const internalFormat = gl.DEPTH_COMPONENT24;
      const border = 0;
      const format = gl.DEPTH_COMPONENT;
      const type = gl.UNSIGNED_INT;
      const data = null;
      gl.texImage2D(gl.TEXTURE_2D, level, internalFormat,
        targetTextureWidth, targetTextureHeight, border, format, type, data);
      // set the filtering so we don't need mips
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
      gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);
      // attach the depth texture to the framebuffer
      gl.framebufferTexture2D(gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
               gl.TEXTURE_2D, depthTexture, level);
}
```

Render Function and Shaders

```
function render() {
     pass1() ;
     pass2() ;
}

function pass1() {
     // render to our targetTexture by binding the framebuffer
     gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
     gl.viewport( 0, 0, targetTextureWidth,
            targetTextureHeight ); // VERY IMPORTANT!!
     //draw scene
     gl.activeTexture(gl.TEXTURE0);
     gl.bindTexture(gl.TEXTURE_2D, textureArray[0].textureWebGL);
     gl.uniform1i(gl.getUniformLocation(program, "texture1"), 0);
     …
}
```

Render Function and Shaders

```
function pass1() {
     // render to our targetTexture by binding the framebuffer
     gl.bindFramebuffer(gl.FRAMEBUFFER, fbo);
     gl.viewport( 0, 0, targetTextureWidth,
            targetTextureHeight ); // VERY IMPORTANT!!
     //draw scene
     gl.activeTexture(gl.TEXTURE0);
     gl.bindTexture(gl.TEXTURE_2D, textureArray[0].textureWebGL);
     gl.uniform1i(gl.getUniformLocation(program, "texture1"), 0);
     …
} // target texture now holds the scene

function pass2() {
    // bind the back buffer by binding null
    gl.bindFramebuffer(gl.FRAMEBUFFER, null);
    gl.viewport( 0, 0, canvas.width, canvas.height ); //VERY IMPORTANT!
    // use the texture that was just rendered as unit 0
    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, targetTexture);
    gl.uniform1i(gl.getUniformLocation(program, "texture1"), 0);
    …
}
```

Vertex Shader for both passes

Standard ADS vertex shader

Fragment Shader for both passes

```
#version 300 es
precision mediump float;
uniform sampler2D texture1;
uniform int useTextures ;

in vec4 fColor;
in vec2 fTexCoord ;
out vec4 fragColor ;

void
main()
{
    vec4 c1;
    c1 = texture( texture1, fTexCoord );
    fragColor = mix(c1,fColor,0.5);

    fragColor.a = 1.0 ;
}
```