Opengl extension

EXT_FRAMEBUFFER_OBJECT

DEUKHYUN CHA

EXT_framebuffer_object (1/15)

- Purpose : enables a form of “offscreen” rendering
- Approved by the ARB "superbuffers" Working Group on January 31, 2005

- “framebuffer-attachable image” (FAI)
  - Newly defined rendering destination in this extension
  - Attaching the FAI to the standard GL logical buffers as one of the standard GL logical buffers: color, depth, and stencil
  - Source and destination of fragment operation
  - Texture can be used as FAI: support “render to texture”
    - By specifying mipmap level(1D, 2D), cube map face(cube map), and z-offset(3D)

- “renderbuffer”
  - Encapsulation a single 2D pixel image which can be used as FAI
  - Rendering to GL logical buffer types which have no corresponding texture format

EXT_framebuffer_object (2/15)

- "framebuffer object"
  - Efficient switching between collections of FAI
  - Containing the status defines the traditional GL framebuffer, including its set of images
    - Window-system defined and managed the collection of images prior to this extension
  - Much simpler to enable rendering to texture than "ARB_render_texture"
  - API contained wholly with in the GL API and has no window-system components

EXT_framebuffer_object (3/15)

- One of old style: “pBuffer”

create

```
int attribList[] =
{
   WGL_RED_BITS_ARB,            32,
   WGL_GREEN_BITS_ARB,          32,
   WGL_BLUE_BITS_ARB,           32,
   WGL_ALPHA_BITS_ARB,          32,
   //float 으로 잡을 경우
   WGL_PIXEL_TYPE_ARB,   WGL_TYPE_RGBA_FLOAT_ATI,
   //declare a number of pbuffer!
   //WGL_AUX_BUFFERS_ARB,     4,
   WGL_DOUBLE_BUFFER_ARB,  true,
  //WGL_FLOAT_COMPONENTS_NV,      true,       //only for NVIDIA texture rectangle
   WGL_DRAW_TO_PBUFFER_ARB,      true,
   WGL_BIND_TO_TEXTURE_RGBA_ARB,  true,   //pBuffer
    0,
};

hWindowDC = wglGetCurrentDC();
hWindowRC = wglGetCurrentContext();

wglChoosePixelFormatARB(hWindowDC, attribList, NULL, 1, &format, &nformats);

handle = wglCreatePbufferARB(hWindowDC, format, width, height, attribList1);

hDC = wglGetPbufferDCARB(handle);

hRC = wglCreateContext(hDC);

wglShareLists(hWindowRC, hRC);
```

activate

```
wglMakeCurrent(hDC, hRC);
```

deactivate

```
wglMakeCurrent(hWindowDC, hWindowRC);
```

use

```
wglReleaseTexImageARB (handle, WGL_FRONT_LEFT_ARB);
glDrawBuffer(GL_FRONT);

wglMakeCurrent(hDC, hRC);

// draw

wglMakeCurrent(hWindowDC, hWindowRC);
```

EXT_framebuffer_object (4/15)

- APIs for binding and managing framebuffer objects
  - void GenFramebuffersEXT(sizei n, uint *ids);
  - void DeleteFramebuffersEXT(sizei n, uint *framebuffers);
  - void BindFramebufferEXT(enum target, uint framebuffer);
    - <target> set to FRAMEBUFFER_EXT and <framebuffer> set to the unused name

- APIs for binding and managing renderbuffer objects
  - void GenRenderbuffersEXT(sizei n, uint *renderbuffers);
  - void DeleteRenderbuffersEXT(sizei n, const uint *renderbuffers);
  - void BindRenderbufferEXT(enum target, uint renderbuffer);
    - <target> set to RENDERBUFFER_EXT and <renderbuffer> set to the unused name
  - void RenderbufferStorageEXT(enum target, enum internalformat, sizei width, sizei height);
    - INVALID_VALUE : if either <width> or <height> is greater than MAX_RENDERBUFFER_SIZE_EXT
    - OUT_OF_MEMORY : if the GL is unable to create a data store of the requested size

EXT_framebuffer_object (5/15)

- API for attaching renderbuffer images to a framebuffer
  - void FramebufferRenderbufferEXT(enum target, enum attachment, enum renderbuffertarget, uint renderbuffer);
    - <target> must be FRAMEBUFFER_EXT
    - <attachment> must be one of the framebuffer attachment points
    - <renderbuffertarget> must be RENDERBUFFER_EXT
    - <renderbuffer> should be set to the name of the renderbuffer object
    - INVALID_OPERATION: if the current value of FRAMEBUFFER_BINDING_EXT is zero

- Framebuffer attachment points
  - COLOR_ATTACHMENT0_EXT ... COLOR_ATTACHMENTn_EXT
    - (where n is from 0 to MAX_COLOR_ATTACHMENTS_EXT-1)
  - DEPTH_ATTACHMENT_EXT
  - STENCIL_ATTACHMENT_EXT

EXT_framebuffer_object (6/15)

- APIs for attaching texture images to a framebuffer
  - void FramebufferTexture1DEXT(enum target, enum attachment, enum textarget, uint texture, int level);
  - void FramebufferTexture2DEXT(enum target, enum attachment, enum textarget, uint texture, int level);
  - void FramebufferTexture3DEXT(enum target, enum attachment, enum textarget, uint texture, int level, int zoffset);
    - <target> must be FRAMEBUFFER_EXT
    - <attachment> must be one of the framebuffer attachment points
    - < textarget> must be a name of textrue object or name of cube map face (TEXTURE_CUBE_MAP_POSITIVE_X, …, TEXTURE_CUBE_MAP_NEGATIVE_Z)
    - INVALID_OPERATION: if the current value of FRAMEBUFFER_BINDING_EXT is zero

EXT_framebuffer_object (7/15)

- General procedure to use framebuffer object
  - ① Generate framebuffer attachable image
    - Texture image for render to texture
    - Renderbuffer object in other case
  - ② Set the properties of generated framebuffer attachable image
  - ③ Generate framebuffer object
  - ④ Bind framebuffer object
  - ⑤ Attach framebuffer attachable image to current framebuffer object
  - ⑥ Bind again when using the framebuffer object
  - ⑦ Draw
  - ⑧ Delete framebuffer object after whole render process done
  - ⑨ Delete framebuffer attachable image

EXT_framebuffer_object (8/15)

- Framebuffer Completeness
  - if it is the window-system-provided framebuffer, or if all the following conditons are true
    - All framebuffer attachment points are "framebuffer attachment complete"
      (error: FRAMEBUFFER_INCOMPLETE_ATTACHMENT_EXT)
    - There is at least one image attached to the framebuffer
      (error: FRAMEBUFFER_INCOMPLETE_ATTACHMENT_EXT)
    - All attached images have the same width and height.
      (error: FRAMEBUFFER_INCOMPLETE_DIMENSIONS_EXT)
    - All images attached to the attachment points COLOR_ATTACHMENT0_EXT through COLOR_ATTACHMENTn_EXT must have the same internal format.
      (error: FRAMEBUFFER_INCOMPLETE_FORMATS_EXT)
    - The value of FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE_EXT must not be NONE for any color attachment point(s) named by DRAW_BUFFERi.
      (error: FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER_EXT)
    - If READ_BUFFER is not NONE, then the value of FRAMEBUFFER_ATTACHMENT_OBJECT_TYPE_EXT must not be NONE for the color attachment point named by READ_BUFFER.
      (error: FRAMEBUFFER_INCOMPLETE_READ_BUFFER_EXT)
    - The combination of internal formats of the attached images does not violate an implementation-dependent set of restrictions.
      (error: FRAMEBUFFER_UNSUPPORTED_EXT)

EXT_framebuffer_object (9/15)

- Mapping between Pixel and Element in Attached Image
  - If the attached image is a renderbuffer image, then the window coordinates (x[w], y[w]) corresponds to the value in the renderbuffer image at the same coordinates
  - If the attached image is a texture image, then the window coordinates (x[w], y[w]) correspond to the texel (i, j, k) as follows:
    - i = (x[w] – b)

      j = (y[w] – b)

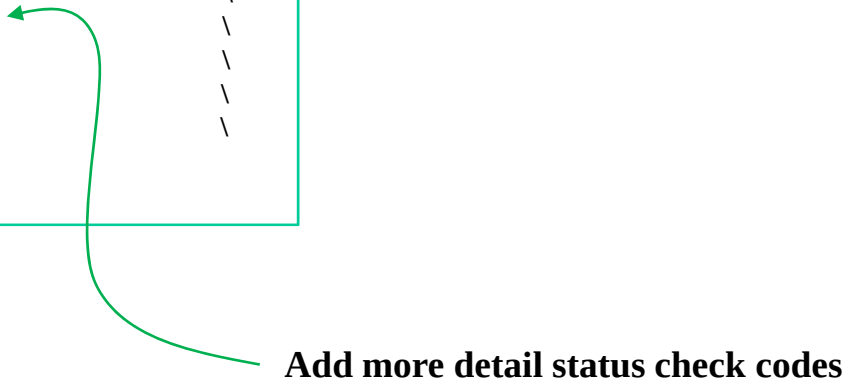
      k = (zoffset - b)

      (b is the texture image's border width, and zoffset is the value of FRAMEBUFFER_ATTACHMENT_TEXTURE_3D_ZOFFSET for the selected logical buffer)

EXT_framebuffer_object (10/15)

- Usage Example #1
  - Framebuffer error check routine

```
#define CHECK_FRAMEBUFFER_STATUS()                                  \
    {                                                               \
     GLenum status;                                                 \
     status = glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);      \
     switch(status) {                                               \
       case GL_FRAMEBUFFER_COMPLETE_EXT:                            \
         break;                                                     \
       case GL_FRAMEBUFFER_UNSUPPORTED_EXT:                         \
         /* choose different formats */                             \
         break;                                                     \
       default:                                                     \
         /* programming error; will fail on all hardware */         \
         assert(0);                                                 \
     }
    }
```



EXT_framebuffer_object (11/15)

- Usage Example #2
  - Render to 2D texture with a depth buffer

```
// Enable render-to-texture
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb);

// Set up color_tex and depth_rb for render-to-texture
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
                    GL_COLOR_ATTACHMENT0_EXT,
                    GL_TEXTURE_2D, color_tex, 0);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
                      GL_DEPTH_ATTACHMENT_EXT,
                      GL_RENDERBUFFER_EXT, depth_rb);

// Check framebuffer completeness at the end of initialization.
CHECK_FRAMEBUFFER_STATUS();

<draw to the texture and renderbuffer>

// Re-enable rendering to the window
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

glBindTexture(GL_TEXTURE_2D, color_tex);
<draw to the window, reading from the color_tex>
```

EXT_framebuffer_object (12/15)

- Usage Example #3 (1/2)
  - Render-to-texture loop with automatic mipmap generation. There are N framebuffers, N mipmap color textures, and a single shared depth renderbuffer. The depth renderbuffer is not a mipmap.

1/3

```
GLuint fb_array[N];
GLuint color_tex_array[N];
GLuint depth_rb;

glGenFramebuffersEXT(N, fb_array);
glGenTextures(N, color_tex_array);
glGenRenderbuffersEXT(1, &depth_rb);

// initialize color textures
for (int i=0; i<N; i++) {
  glBindTexture(GL_TEXTURE_2D, color_tex_array[N]);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB8, 512, 512, 0,
          GL_RGB, GL_INT, NULL);

  // establish a mipmap chain for the texture
  glGenerateMipmapEXT(GL_TEXTURE_2D);
}
```

2/3

```
// initialize depth renderbuffer
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depth_rb);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT,
                GL_DEPTH_COMPONENT24, 512, 512);

// setup framebuffers, sharing depth
for (int i=0; i<N; i++) {
  glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb_array[i]);
  glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT,
                GL_COLOR_ATTACHMENT0_EXT,
                GL_TEXTURE_2D, color_tex_array[i], 0);
  glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
                GL_DEPTH_ATTACHMENT_EXT,
                GL_RENDERBUFFER_EXT, depth_rb);
}
```

EXT_framebuffer_object (13/15)

- Usage Example #3 (2/2)
  - Render-to-texture loop with automatic mipmap generation. There are N framebuffers, N mipmap color textures, and a single shared depth renderbuffer. The depth renderbuffer is not a mipmap.

3/3

```
// Check framebuffer completeness at the end of initialization.
CHECK_FRAMEBUFFER_STATUS();

loop {
   glBindTexture(GL_TEXTURE_2D, 0);

   for (int i=0; i<N; i++) {
     glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb_array[i]);
     <draw to texture i>
   }

   glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

   // automatically generate mipmaps
   for (int i=0; i<N; i++) {
     glBindTexture(GL_TEXTURE_2D, color_tex_array[i]);
     glGenerateMipmapEXT(GL_TEXTURE_2D);
   }

   <draw to the window, reading from the color textures>
}
```

EXT_framebuffer_object (14/15)

- Usage Example #4 (1/2)
  - Use cube map by framebuffer attachable image

1/2

```
GLuint fb_array;
GLuint cube_tex_array;
GLuint depth_rb;

glGenFramebuffersEXT(1, &fb_array);
glGenTextures(1, &cube_tex_array);
glGenRenderbuffersEXT(1, &depth_rb);

glBindTexture(GL_TEXTURE_CUBE_MAP_ARB, cube_tex_array);
glTexParameteri(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);

for(int i=0;i<6;i++) {
    //create cube map texture
   glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB+i, 0, TEXTURE_INTERNAL_TYPE,
                cubeTexW, cubeTexH, 0, GL_RGBA, GL_FLOAT, NULL);
}

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb_array);
for(int i=0; i<6; i++) {
  glFramebufferTexture2DEXT( GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                             GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB+i, cube_tex_array, 0);
 }

glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depth_rb);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT32, cubeTexW, cubeTexH);
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depth_rb);
```

EXT_framebuffer_object (15/15)

- Usage Example #4 (2/2)
  - Use cube map by framebuffer attachable image

2/2

```
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fb[0]);

for(int i=0; i<6; i++) {
  glFramebufferTexture2DEXT( GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                             GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB+i, cube_tex_array, 0);
  glDrawBuffer(GL_COLOR_ATTACHMENT0_EXT);
  <draw>
}

// bind cube map
glBindTexture(GL_TEXTURE_CUBE_MAP_ARB, cube_tex_array);
```