NVIDIA®

OpenGL Render-to-Texture

Chris Wynn

NVIDIA Corporation

What is Dynamic Texturing?

- **The creation of texture maps “on the fly” for use in real time.**

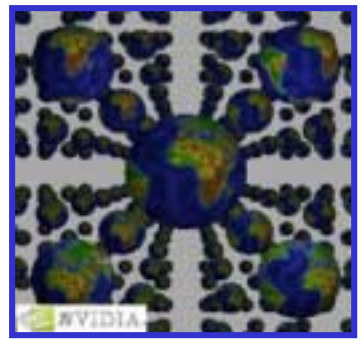
- **Simplified view:**

  Loop:

  1. **Render an image.**
  2. **Create a texture from that image.**
  3. **Use the texture as you would a static texture.**

Applications of Dynamic Texturing

- Impostors
- Feedback Effects
- Dynamic Cube / Environment map generation
- Dynamic Normal map generation
- Dynamic Volumetric Fog
- Procedural Texturing
- Dynamic Image Processing
- Physical (PDE) Simulation



Challenge of Dynamic Texturing

- **Performance Bottlenecks**

- **Simplified view:**

  Loop:

  1. **Render an image.**
  2. **Create a texture from that image.**
  3. **Use the texture as you would a static texture.**

***Step 2 is primary bottleneck but 1 and 3 can be relevant as well.**

Methods for Creating the Texture

How to get rendered image into a texture?

- **glReadPixels() → glTexImage*() ?**
  - **Slow.**
- **glCopyTexImage*()**
  - **Better.**
- **glCopyTexSubImage*()**
  - **Even Better.**
- **Render Directly to Texture**
  - **Eliminates “texture copy” – potentially optimal**

Rendering Directly to a Texture

- Not a core part of OpenGL 1.3, but ARB extensions make this possible on most GPUs.
- Required Extensions:
  - WGL_ARB_extensions_string
  - WGL_ARB_render_texture
  - WGL_ARG_pbuffer
  - WGL_ARB_pixel_format
- Available on all NVIDIA products since GeForce (requires 28.40 driver)

Rendering Directly to a Texture: An Overview

Basic Idea: Allow a p-buffer to be bound as a texture

- Create a texture object
- Create a “Render Texture” (i.e. the pbuffer)
- Loop as necessary:
  - Make the pbuffer the current rendering target
  - Render an image
  - Make the window the current rendering target
  - Bind the pbuffer to the texture object
  - Use the texture object as you would any other
  - Release the pbuffer from the texture object
- Clean Up

Creating the Texture Object

Just as you would for a regular texture -- no need to specify the actual texture data

```
// Create a render texture object
glGenTextures( 1, &render_texture );
glBindTexture( GL_TEXTURE_2D, render_texture );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                 GL_LINEAR_MIPMAP_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_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 );
```

Creating the Pbuffer

- Quick Overview
  1. Get a valid device context

     HDC hdc = wglGetCurrentDC();

  2. Choose a pixel format

     Specify a set of minimum attributes
     - Color bits, Depth bits, Stencil bits, single/double, etc.
     - Must also specify WGL_DRAW_TO_PBUFFER and either WGL_BIND_TO_TEXTURE_RGB_ARB or WGL_BIND_TO_TEXTURE_RGBA_ARB as TRUE.

     Then call wglChoosePixelFormat()
     - Returns a list of formats which meet minimum requirements.
     - fid = pick any format in the list.

Creating the Pbuffer (cont.)

3. Create the pbuffer

HPBUFFER hbuf = wglCreatePbufferARB(hdc, fid, width, height, attr);

width and height are dimensions of the pbuffer

“attr” is a list of other properties for your pbuffer.

- **Set WGL_TEXTURE_FORMAT_ARB:**
  - **WGL_TEXTURE_RGB_ARB or WGL_TEXTURE_RGBA_ARB**
- **Set WGL_TEXTURE_TARGET_ARB:**
  - **WGL_TEXTURE_1D_ARB, WGL_TEXTURE_2D_ARB, or WGL_TEXTURE_CUBE_MAP_ARB**
- **Set WGL_MIPMAP_TEXTURE_ARB to non-zero value to request space for mipmaps.**
- **Set WGL_PBUFFER_LARGEST_ARB to non-zero value to obtain largest possible pbuffer.**

Creating the Pbuffer (cont.)

4. Get the device context for the pbuffer

```
hpbufdc = wglGetPbufferDCARB( hbuf );
```

5. Get a rendering context for the pbuffer:
   - Create a new one – pbuffer gets its own GL state:

```
hpbufglrc = wglCreateContext( hpbufdc );
```

6. Determine the actual dimension of the created pbuffer

```
wglQueryPbufferARB( hbuf,
                    WGL_PBUFFER_WIDTH_ARB, width );
wglQueryPbufferARB( hbuf,
                    WGL_PBUFFER_HEIGHT_ARB, height );
```

Rendering to the Texture

- Can be done anytime once the creation of the pbuffer is complete
  - Initialization function, display loop, every 10 frames, etc.
- Must make the rendering context for the pbuffer current using wglMakeCurrent:

```
wglMakeCurrent( hpbufdc, hpbufglrc );
Issue OpenGL drawing commands
wglMakeCurrent( hwindc, hwinglrc );
```

Rendering to the Texture (cont.)

- The render to texture mechanism allows for rendering to specific regions of a texture:
  - A specific level of a mipmapped texture
  - A specific face of a cube map texture
  - A specific mip level of a specific face of a cube map texture

- Can use wglSetPbufferAttribARB() to choose which cube map face or mipmap level to render.

```
BOOL wglSetPbufferAttribARB (HPBUFFERARB hPbuffer,
                             const int *piAttribList)
```

Binding the pbuffer to the texture object

After binding the texture object…

Call wglBindTexImageARB to bind the pbuffer to the texture object.

```
BOOL wglBindTexImageARB ( HPBUFFERARB hPbuffer,
                          int iBuffer )
```

Set <iBuffer> to WGL_FRONT_LEFT_ARB or WGL_BACK_LEFT_ARB depending upon which buffer was used for rendering the texture.

Releasing the pbuffer from the texture object

***** You *must* release the pbuffer from the texture before you can render to it again. *****

Call wglReleaseTexImageARB to release the color buffer of the pbuffer.

```
BOOL wglReleaseTexImageARB ( HPBUFFERARB hPbuffer,
                             int iBuffer)
```

Clean Up

When finished using a render texture, it is important to safely release the resources consumed by the pbuffer.

3 Step Process

1. Delete the rendering context
2. Release the Pbuffer's device context
3. Destroy the Pbuffer

```
wglDeleteContext( hpbufglrc );
wglReleasePbufferDCARB( hbuf, hpbufdc );
wglDestroyPbufferARB( hbuf );
```

Automatic Mipmap Generation

WGL_ARB_render_texture allows for rendering to a specific mipmap level of mipmapped texture.

- **Requires render pass per mipmap level**

Improve speed of generating mipmap levels using GL_SGIS_generate_mipmap extension.

Allows hardware to auto-generate mipmap levels of the render texture whenever the base level 0 is updated.

- **One render pass builds entire mipmap pyramid**
- **Fast, easy to use:**

```
glTexParameteri( GL_TEXTURE_2D,
                 GL_GENERATE_MIPMAP_SGIS, GL_TRUE );
```

Non-Power-of-Two Textures

OpenGL only supports textures with 2^m x 2^n resolution.

But "Non-Power-of-Two" textures can be useful

- **Matching some screen resolution or bounding region that is not necessarily a power of two (800x600)**

Restriction lifted w/ WGL_NV_render_texture_rectangle extension:

- **Non-power-of-two textures BUT some differences**
- **Texture coordinates map differently**
  - **s,t range: [0,Width], [0,Height] respectively instead of usual [0,1], [0,1] range.**
- **No mipmap filtering.**
- **No border texels or repeat texture wrap modes supported**

Non-Power-of-Two Textures

During the pbuffer creation process for a texture rectangle render texture, be sure to specify:

WGL_BIND_TO_TEXTURE_RECTANGLE_RGB[A]_NV as TRUE when choosing the pixel format (step 2)

And

WGL_TEXTURE_TARGET_ARB as WGL_TEXTURE_RECTANGLE_NV when creating the pbuffer (step 3)

Depth Textures

OpenGL supports “depth” textures via the GL_SGIX_depth_texture extension.

Used with Shadow-Mapping (see GL_SGIS_Shadow_map)

To setup a pbuffer for directly rendering to a depth texture must use WGL_NV_render_depth_texture extension

Depth Textures

During the pbuffer creation process for a depth texture render texture, be sure to specify:

WGL_BIND_TO_TEXTURE_DEPTH_NV as TRUE when choosing the pixel format (step 2)

And

WGL_DEPTH_TEXTURE_FORMAT_NV as WGL_TEXTURE_DEPTH_COMPONENT_NV when creating the pbuffer (step 3)

Single Pbuffer for Multiple Textures: Color and Depth

Can share a single Pbuffer between 2 textures:

WGL_BIND_TO_TEXTURE_DEPTH_NV as TRUE and

WGL_BIND_TO_TEXTURE_RGB[A]_ARB as TRUE when choosing the pixel format (step 2)

Also

WGL_TEXTURE_FORMAT_ARB as WGL_TEXTURE_RGB[A]_ARB and

WGL_DEPTH_TEXTURE_FORMAT_NV as WGL_TEXTURE_DEPTH_COMPONENT_NV when creating the pbuffer (step 3)

Bind to 2 textures – a depth texture and a color texture

For More Information…

- Questions to: cwynn@nvidia.com
- NVIDIA Developer Website
  - http://www.nvidia.com/developer
  - Pbuffer and Render-to-Texture Whitepapers and Demos.