Using P-Buffers for Off-Screen Rendering

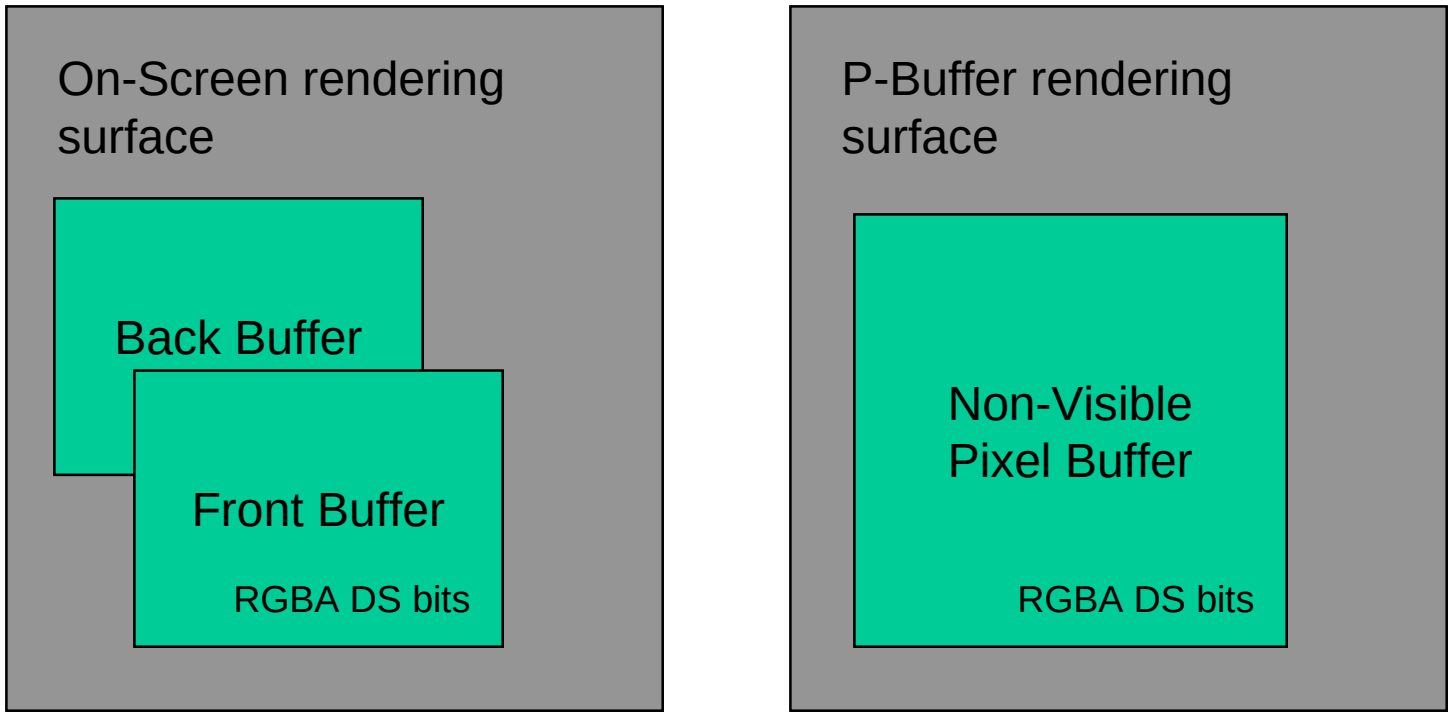
Chris Wynn

GAMEDevelopers CONFERENCE 2001

What is a P-Buffer?

- Off-screen pixel buffer
  - Standard properties (RGBA bits, Z-bits, stencil, etc.) – same as on-screen buffer (window)
- Used for generating intermediate images
  - Shadow Map image
  - Dynamic texture image
- Particularly useful when on-screen image properties are too constraining
  - Dimension (width, height)
  - Properties (pixel format)
  - Occlusion (pixel ownership test)

What is a P-Buffer?



For On-Screen rendering surface: buffer dimensions and bit properties are constrained by the current display mode.

For P-Buffer rendering surface: dimensions and bit properties are independent of the current display mode.

Using P-Buffers

- Windows
  - WGL_ARB_pixel_format
  - WGL_ARB_pbuffer
- Linux
  - Supported in GLX 1.3
- MAC
  - Future extension(s)

Using P-Buffers

- Setting up P-Buffers can be tedious ☹
  - Requires windowing system specific calls
  - Can be “abstracted” away ☺
    - Implement once and reuse!
    - Something like glutInitWindowSize() and glutInitDisplayString()/glutInitDisplayMode()
- Three Key Components – same as for a window
  - Creating a P-Buffer
  - Binding a P-Buffer
  - Destroying a P-Buffer

P-Buffer Creation (On Windows)

- **Quick Overview**

1. **Get a valid device context**

   HDC hdc = wglGetCurrentDC();

2. **Choose a pixel format**

   Specify a set of minimum attributes

   - **Color, Depth, Stencil bits, etc.**

   Then call wglChoosePixelFormat()

   - **Returns a list of formats which meets minimum requirements.**
   - **fid = pick any format in the list.**

P-Buffer Creation (On Windows)

- **Quick Overview (cont.)**

3. **Create the pbuffer**

   HPBUFFER hbuf = wglCreatePbufferARB(hdc, fid, w, h, attr);

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

   Can use to obtain largest possible buffer if w x h not available.

4. **Get the device context for the pbuffer**

   hpbufdc = wglGetPbufferDCARB(hbuf);

5. **Create a rendering context for the pbuffer**

   hpbufglrc = wglCreateContext(hpbufdc);

Binding a P-Buffer (On Windows)

Easy ☺

wglMakeCurrent(hpbufdc, hpbufglrc);

- **Makes the P-Buffer's rendering context current**
- **Subsequent OpenGL primitives rendered to the off-screen buffer**

To make the visible buffer the current context, call wglMakeCurrent() with the device context and rendering context for the visible window.

Destroying a P-Buffer (On Windows)

3 Step Process

1. **Delete the rendering context**
2. **Release the P-buffer's device context**
3. **Destroy the P-buffer**

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

Handling a Mode-Switch (On Windows)

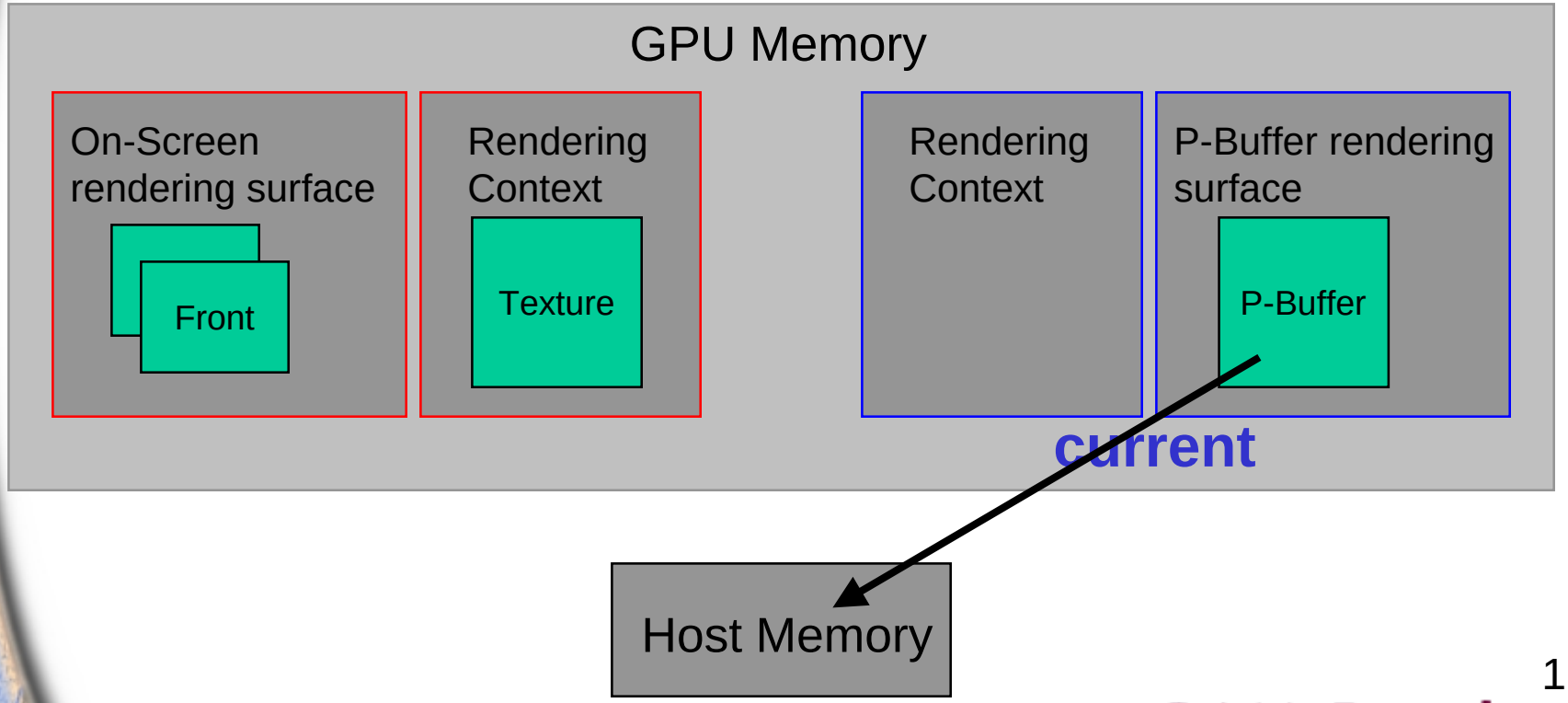
- **Memory associated with a P-buffer not guaranteed to remain valid when a "mode-switch" occurs i.e. ChangeDisplayMode()**

- **After a mode-switch, query if the P-buffer is still valid:**

```
int flag;
wglQueryPbufferARB( hbuf, WGL_PBUFFER_LOST_ARB, &flag );
if ( flag != 0 ) {
    Destroy P-Buffer
    Recreate P-Buffer
}
```

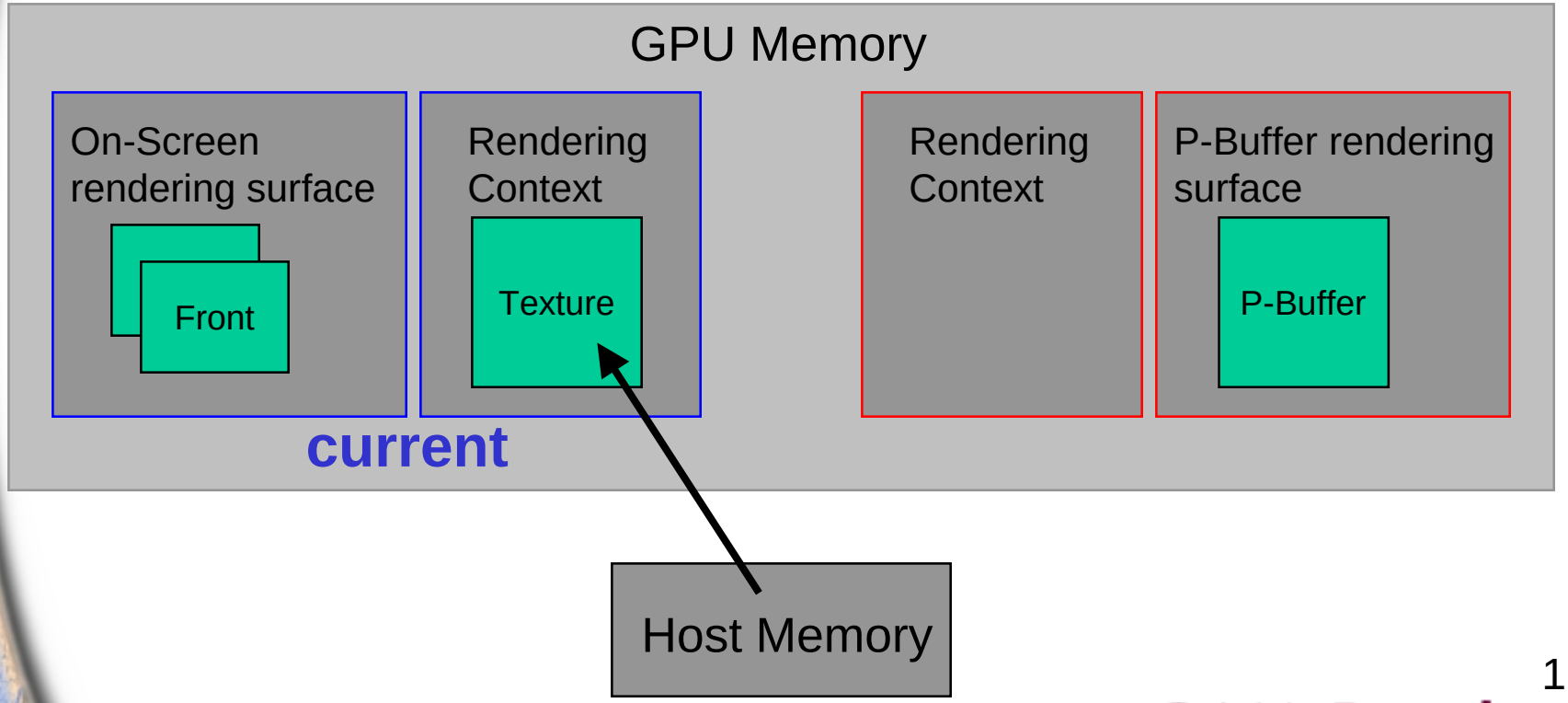
Retrieving Data from a P-Buffer

- **Round trip to host memory… (step 1)**
  - **Read to Host Memory**



Retrieving Data from a P-Buffer

- **Round trip to host memory… (step 2)**
  - **Write to texture Memory**

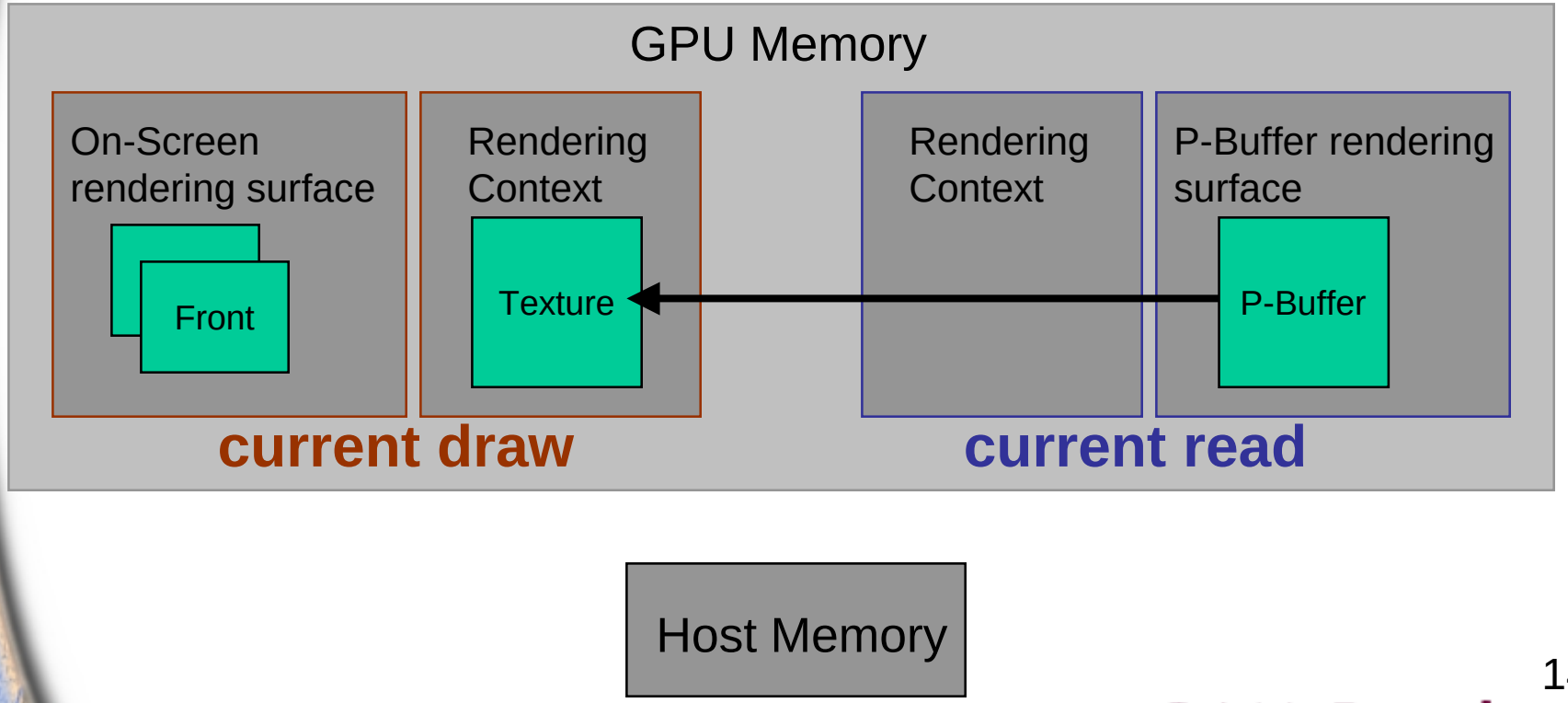


Retrieving Data from a P-Buffer

- **Round trip to host memory (Implementation)**
  - **Bind to P-Buffer**
  - **Render to P-Buffer**
  - **glReadPixels()**
  - **Bind to on-screen rendering surface**
  - **glTexSubImage2D()**
  - **Render frame**

Retrieving Data from a P-Buffer

- **Two “GPU-local” Scenarios**
  - **Copy-to-Texture via “make current read” extension**

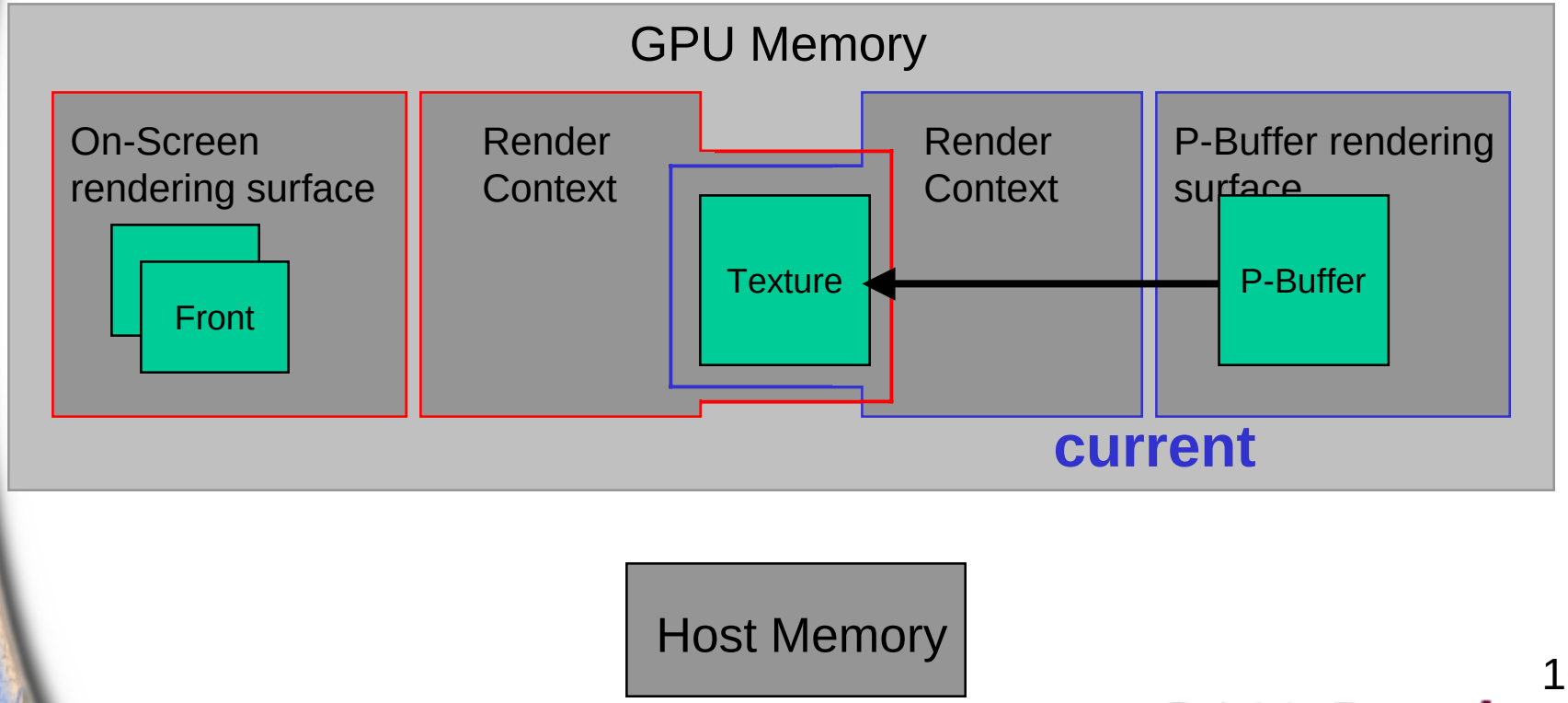


Retrieving Data from a P-Buffer

- Two “GPU-local” Scenarios
  - Copy-to-Texture via “make current read”
    - Use WGL_ARB_make_current_read extension to bind separate draw and read contexts
      - wglMakeContextCurrentARB(hDrawDC, hReadDC, hglrc)
    - Bind to P-buffer
    - Render to P-buffer
    - wglMakeContextCurrentARB(hvisdc, hbufdc, hvisglrc);
    - glCopyTexSubImage();
    - Render frame

Retrieving Data from a P-Buffer

- **Two “GPU-local” Scenarios**
  - **Copy-to-Texture via “shared textures”**



Retrieving Data from a P-Buffer

- Two “GPU-local” Scenarios
  - Copy-to-Texture via “shared textures”
    - Use wglShareLists(hvisglrc, hpbufglrc)
      - Allows sharing of ALL display list and texture objects between rendering contexts.
      - Call just once immediately after creating the P-buffer.
    - Bind to P-buffer
    - Render to P-buffer
    - glCopyTexSubImage2D();
    - Bind to on-screen rendering surface
    - Render frame

P-Buffers: On NVIDIA GPUs

- **Windows**
  - **Hardware accelerated for TNT, TNT2, and the ENTIRE GeForce family of GPUs.**
  - **Release 10 driver and beyond**
- **Linux and MAC support coming…**

Things to Keep in Mind…

- **P-Buffers consume Video Memory**
  - **Frame Buffer, Textures, Display Lists, an P-Buffers in same memory.**
  - **Large/Lots of P-Buffers on low-end may limit performance**
    - **One P-Buffer is often enough**
    - **Error check the creation routines**

For More Information…

- **Extensions Registry**
  - http://www.oss.sgi.com/projects/ogl-sample/registry/
  - **Details how the Windows and SGI P-Buffer extensions work from an Implementer's point-of-view**
- **NVIDIA Developer Website**
  - **Technical Paper: "Using P-Buffers for Off-Screen Rendering in OpenGL"**
    - **Details how to use P-Buffers on Windows**
  - **Technical Demos**
    - **Illustrates how to use P-Buffers on Windows**
    - **Easy-to-Use C++ P-Buffer abstraction**
      - **3 minutes to P-Buffer enabled code!!!**

Questions, comments, feedback?

- **Chris Wynn, cwynn@nvidia.com**
- **www.nvidia.com/Developer**