What is Dynamic Texturing?

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

- A simplified view:
  - Loop:
    - Render an image.
    - Create a texture from that image.
    - 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
Overview

- Copying Texture Data
- Off-Screen Rendering with Pixel Buffers
- Rendering Directly To Textures
- Procedural Texturing
The Basics: Copying Texture Data

• How do we get a rendered image into a texture?
  ▪ `glReadPixels()` → `glTexImage*(())` ?
    ▪ SLOW!
  ▪ `glCopyTexImage*(())`
    ▪ Better.
  ▪ `glCopyTexSubImage*(())`
    ▪ Best (currently).
• Render to Texture
  ▪ Coming Soon!
glCopyTexSubImage

- Not just for sub-images anymore!

- Performance is better than glCopyTexImage
  - Doesn’t require allocation of texture memory.
  - Optimized in NVIDIA’s Release10 driver.
What About Mipmaps?

- Sometimes we want mipmaps for our dynamic textures.

- How do we generate them?
  - The obvious way: generate them yourself.
  - GluBuild2DMipmaps().
  - Automatic mipmap generation.
Automatic Mipmap Generation!

- **SGIS_generate_mipmap extension**
  - New token `GL_GENERATE_MIPMAP_SGIS` for `glTexParameteri()`
  - Set to `GL_TRUE`, causes mipmap levels to be updated anytime base level image changes
  - Faster than `gluBuild2DMipmaps`

```c
glBindTexture( GL_TEXTURE2D, tid );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_NEAREST );
glTexParameteri( GL_TEXTURE_2D, GL_GENERATE_MIPMAP_SGIS, GL_TRUE );
glCopyTexSubImage2D( GL_TEXTURE_2D, ... );
```
Automatic Mipmap Generation: On NVIDIA GPUs

- Works with glTexImage[Sub]Image, glCopyTexImage[Sub]Image.
- Extension supported for **ALL** texture formats for **ENTIRE** GeForce family.
- Only HW-Accelerated when used with glCopyTexImage[Sub]Image2D and the following formats:
  - GL_RGB8
  - GL_RGBA8
  - GL_RGB5
- Copies w/ auto-mipmap enabled will copy at 50% the speed of just updating the base level texture.
  - Copies 5x faster with release 10 driver
Off-Screen Rendering with Pixel Buffers

- We don’t always want to use the frame buffer to render our dynamic textures.

- Why not?
  - Resolution is limited to the window resolution.
  - Might need a different pixel format.
  - Can require a lot of OpenGL state juggling.
  - Overlapping windows can mess up copies.
  - Can’t be used to render to texture (more later).

- Use a pbuffer instead!
What is a Pbuffer?

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

For **pbuffer rendering surface**: dimensions and bit properties are independent of the current display mode.
Using Pbuffers

- Windows
  - WGL_ARB_pixel_format extension
  - WGL_ARB_pbuffer extension

- Linux
  - Supported in GLX 1.3

- MAC
  - Future extension(s)
Using Pbuffers

• Setting up pbuffers 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 pbuffer
  • Binding a pbuffer
  • Destroying a pbuffer
Pbuffer Creation (In Windows)

- Quick Overview
  1. Get a valid device context
     
     ```cpp
     HDC hdc = wglGetCurrentDC();
     ```
  2. Choose a pixel format
     
     Specify a set of minimum attributes
     - Color, Depth, Stencil bits, etc.
     - Can specify single- or double-buffered, just like a window.
     - Will usually need only single buffer (save RAM!).
     
     Then call `wglChoosePixelFormat()`
     - Returns a list of formats which meet minimum requirements.
     - `fid = pick any format in the list.`
Pbuffer Creation (In 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.
  4. Get the device context for the pbuffer
    ```
    hdc = wglGetPbufferDCARB( hbuf );
    ``
  5. Get a rendering context for the pbuffer:
    - Either create a new one (pbuffer gets its own GL state!):
      ```
      hglrc = wglCreateContext( hdc );
      ``
    - Or use the current context:
      ```
      hglrc = wglGetCurrentContext();
      ```
Binding a Pbuffer (In Windows)

- Easy!
  
  `wglMakeCurrent( hdc, hglrc );`

  - Makes the pbuffer device context the current rendering target for the rendering context.
  - Subsequent OpenGL primitives rendered to the off-screen buffer.
Destroying a Pbuffer (In Windows)

- 3 Step Process
  1. Delete the rendering context
  2. Release the pbuffer’s device context
  3. Destroy the pbuffer

  `wglDeleteContext( hpbufgIrc );`
  `wglReleasePbufferDCARB( hbuf, hbufdc );`
  `wglDestroyPbufferARB( hbuf );`
Retrieving Data from a Pbuffer

- Copy-to-Texture via “shared textures”
Copy-to-Texture via “shared textures”

- Use wglShareLists( hVisibleGLRC, hPbufferGLRC )
  - Allows sharing of ALL display list and texture objects between rendering contexts.
  - Call just once immediately after creating the Pbuffer.
  - Don’t need if pbuffer uses same GLRC as app window.

- Bind to pbuffer
- Render to pbuffer
- glCopyTexSubImage2D();
- Bind to on-screen rendering surface
- Render frame
Pbuffers: 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…

- **Pbuffers consume Video Memory**
  - Frame Buffer, Textures, Display Lists, and pbuffers all in video memory.
  - Large/Lots of pbuffers on low-end may limit performance
    - One single-buffered pbuffer is often enough.
    - Don’t request depth if you don’t need it.
    - Error check the creation routines.

- **Keep track of your state!**
  - Don’t get confused about which context is current when setting GL state.
Example uses for pbuffers

- Shadow Map Creation.
- Rendering dynamic text to a texture.
- “Pre-baked” terrain texturing:
  - Each terrain vertex has a set of weights for blending basis textures:
    - $w_0 \cdot \text{grass} + w_1 \cdot \text{rocks} + w_2 \cdot \text{dirt} + w_3 \cdot \text{water} + \ldots + \ldots$
  - Pre-blend textures using reg. Combiners and/or multipass into a single texture for each terrain region.
  - Copy to texture using auto-mipmap generation.
  - Use to texture terrain.
Rendering Directly to Textures

- Our most requested OpenGL feature.
- We’re finally going to have it! (in windows)
  - Will be available in an upcoming driver release.
- Implementation of WGL_ARB_render_texture extension.
  - Allows a pbuffer to be bound as a texture.
  - Defines three new functions:
    - wglBindTexImageARB ()
    - wglReleaseTexImageARB ()
    - wglSetPbufferAttribARB ()
Using WGL_ARB_render_texture

1. Create a pbuffer with appropriate pixel format.
   - In wglChoosePixelFormat():
     * Specify WGL_DRAW_TO_PBUFFER and either
       WGL_BIND_TO_TEXTURE_RGB_ARB or
       WGL_BIND_TO_TEXTURE_RGBA_ARB as TRUE.
   - In wglCreatePbufferARB():
     * Set WGL_TEXTURE_FORMAT_ARB:
       * WGL_TEXTURE_RGB_ARB or WGL_TEXTURE_RGBA_ARB
     * Set WGL_TEXTURE_TARGET_ARB:
       * WGL_TEXTURE_CUBE_MAP_ARB, WGL_TEXTURE_1D_ARB, or
         WGL_TEXTURE_2D_ARB
     * Use WGL_MIPMAP_TEXTURE_ARB to request space for mipmaps.
       * If non-zero and the texture format is WGL_TEXTURE_RGB[A]_ARB, then
         storage for mipmaps will be allocated.
     * Set pbuffer width and height to size of the level zero mipmap image.
2. Create a context for the pbuffer.
   - Make the context current to the pbuffer and initialize the context’s attributes.

3. Render to the pbuffer.

4. Make the context current to the window
   - Bind a texture object to the appropriate texture target and set desired texture parameters.
Using WGL_ARB_render_texture

5. Call wglBindTexImageARB to bind the pbuffer drawable to the texture.
   • Set <iBuffer> to WGL_FRONT or WGL_BACK depending upon which color buffer was used for rendering the texture.

```c
BOOL wglBindTexImageARB (HPBUFFERARB hPbuffer, int iBuffer)
```
Using WGL_ARB_render_texture

6. Render to the window using the texture.
7. Call wglReleaseTexImageARB to release the color buffer of the pbuffer. Goto step 3 to generate more frames.

BOOL wglReleaseTexImageARB (HPBUFFERARB hPbuffer, int iBuffer)

*** NOTE: you must release the pbuffer from the texture before you can render to it again. ***
Rendering Cube Maps and Mipmaps

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

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

- Combine dynamic texture creation and programmable shading for endless possibilities!
  - Dynamic bump and normal maps
  - Feedback Effects:
    - Fire, blur, etc.
  - Computation:
    - Cellular Automata
    - Physics
Procedural Texturing Concepts

- Rendering to texture (already discussed).
- Sampling a Texel’s Neighbors.
  - Use vertex programs and register combiners.
- Use of texture shaders:
  - Dependent texture reads, dot products, and other operations.
- Use of Register Combiners:
  - Weighted texture sampling.
Sampling a Texel’s Neighbors

- Very powerful and important technique!
  - The key to using texture ops for SIMD computation.
  - Can think of it as communication between processor elements.
- Offset texture coordinates by a multiple of the texel dimensions.
  - Ideal candidate for a vertex program.
- Initialize offsets based on dimensions of texture:
  - float texelWidth = 1.0f / (float)textureWidth;
  - float texelHeight = 1.0f / (float)textureHeight;
Sampling a Texel’s Neighbors

- **Example:** sampling 4 nearest neighbors.

- **Load the offsets into VP constant memory:**
  
  ```
  glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, 40, -texelWidth, 0, 0, 0); // left
  glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, 41, texelWidth, 0, 0, 0); // right
  glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, 42, 0, texelHeight, 0, 0); // top
  glProgramParameter4fNV(GL_VERTEX_PROGRAM_NV, 43, 0, -texelHeight, 0, 0); // bottom
  ```

- **Render a quad which exactly covers the render buffer with texture coordinates from (0,0) to (1,1)**
Sampling a Texel’s Neighbors

- **Vertex Shader writes different texture coordinates to each texture stage**
- **Each of the four coordinates is offset by vector in constant memory: c[40], c[41], c[42], or c[43].**
- **In a vertex program, add offsets to input texture coordinates, creating 4 sets of independent texture coordinates:**

  
  \[
  \begin{align*}
  &\text{ADD } o[\text{TEX0}], c[40], v[\text{TEX0}]; \\
  &\text{ADD } o[\text{TEX1}], c[41], v[\text{TEX0}]; \\
  &\text{ADD } o[\text{TEX2}], c[42], v[\text{TEX0}]; \\
  &\text{ADD } o[\text{TEX3}], c[43], v[\text{TEX0}];
  \end{align*}
  \]
Sampling a Texel’s Neighbors

- Use register combiners to combine the samples.
- Bind same texture to all four inputs.
- Example nvparse RC1.0 script to average four samples:

```plaintext
const0 = (0.25, 0.25, 0.25, 0.25);
{
  rgb
  {
    discard = tex0 * const0;
    discard = tex1 * const0;
    spare0 = sum();
  }
}
{
  rgb
  {
    discard = tex2 * const0;
    discard = tex3 * const0;
    spare1 = sum();
  }
}
out.rgb = spare0 + spare1;
out.a = spare1.a;
```
Procedural Texturing Example

- Fire effect using feedback:
  - Blur and scroll upward
    - by sampling and averaging neighbors with downward offset.
  - Drive flames with a “seed” texture of bright embers.
Detailed Example: Game Of Life

- **Cellular Automata**
  - Useful for generating noise and other animated patterns to use in blending.
  - The Game Of Life is used as the “embers” texture in the fire demo!

- **Game Of Life demo:**
  - Uses three rendering passes per generation.
  - Dependent texture address texture shader.
  - Register combiners / vertex program to sample all 8 neighbors of a texel.
Rules of the Game of Life

- A cell will be “alive” in the next generation if:
  - The cell is alive in the current generation and has two or three living neighbors, or
  - The cell is not alive in the current generation and has exactly three neighbors.
The Game Of Life

• How the Game of Life demo operates:

Green Channel
Neighbor Count

Blue Channel
On / Off State

Dependent GB texture address operation
The Game Of Life

• Pass One: discard all but blue.
  • Also bias a little to ensure correct addressing.

• Pass Two: count neighbors of each texel.
  • add count to green channel.

• Pass Three: Determine next generation:
  • use sum of passes one and two as input for dependent GB address lookup into rules texture.
The Game Of Life

- How do we count the neighbors of a texel?
  - Use the neighbor sampling from before, slightly modified.
  - Instead of sampling like this:
  - Change offsets to sample all 8 neighbors:
    - Must enable linear texture filtering.
The Game Of Life

- Use results of first two passes to do a dependent lookup into the “rules texture”.
- Value in blue channel acts as t-coordinate.
  - Encodes “cell is alive in current generation”.
- Value in green channel acts as s-coordinate.
  - Encodes number of living neighbors of each cell.

<table>
<thead>
<tr>
<th>Number of Neighbors</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7</td>
</tr>
</tbody>
</table>

Alive / Dead Status

Result is either black or white.
More complex Procedural Shading

- Game of Life is just a simple example.
  - Possibilities are endless!
- Can use texture operations to do physical simulation!
  - Dynamic bump-mapped waves.
  - Neighbor sampling allows finite-difference integration of simple PDEs!
- Demo maintains 3 textures:
  - Force, velocity, and height.
  - Neighbor sampling determines force.
  - Force applied to velocity.
  - Velocity applied to height.
For More Information…

- Questions to: jspitzer@nvidia.com

- NVIDIA Developer Website
  - Pbuffer and auto mipmap gen presentations.
  - Performance presentation.
  - Texture shader and Vertex program presentations.
  - Demos, demos, demos.
    - GL Game Of Life demo.
    - Several D3D Demos using procedural textures.
    - More always coming.