Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Module 9C
Advanced Texturing

Advanced Texturing Techniques

- Material mapping / gloss mapping
  - Alpha mapping
- Bump mapping
  - Blinn's method
  - Normal mapping
  - Relief mapping
- Environment mapping
  - Blinn and Newell's method
  - Sphere mapping
  - Cubic environment mapping
- Light mapping
- Image based effects
  - Rendering spectrum
  - Skyboxes
  - Billboarding
- Texturing with Shaders
  - Procedural textures with shaders

Multipass Rendering

- **Multipass rendering:**
  - Evaluate separate parts of the illumination equation in separate passes
    - Each pass modifies previous results
    - Framebuffer is used to store intermediate results
  - Prior to shaders: hardware accelerators were not flexible enough to solve complex lighting models in a single pass
  - Example: Modulate diffuse color by a texture but not the specular highlight
    - Use 2 passes: first add diffuse component with texture modulation, then in 2nd pass add specular component without texture
- Quake III used up to 10 passes
  - Passes 1-4: Accumulate bump map
  - Pass 5: diffuse lighting
  - Pass 6: base texture
  - Pass 7: specular lighting
  - Pass 8: emissive lighting
  - Pass 9: volumetric/atmospheric effects
  - Pass 10: screen flashes
  - On less capable hardware might only perform Pass 5 and 6

Multipass Rendering

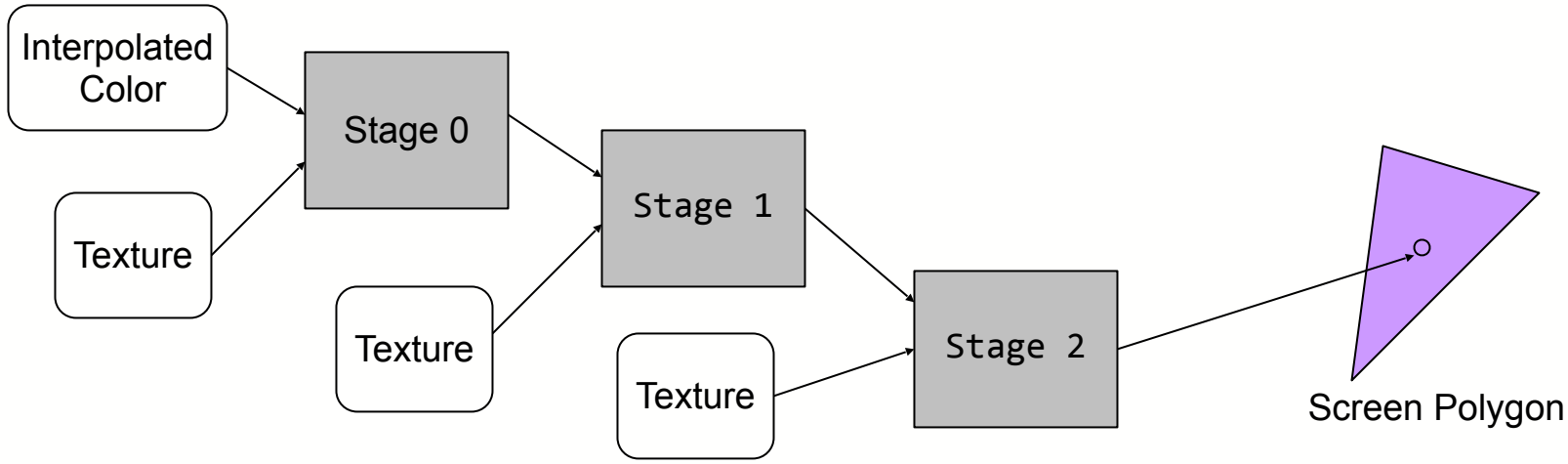
- After application of texture in the graphics pipeline a **fragment** is produced for the pixel
  - Color (RGBA) and geometric information (z-depth, stencil value)
  - Normally the fragment replaces the pixel in the framebuffer
    - Or is discarded if fails z buffer or stencil test
- With multipass rendering the fragment is combined with the current pixel
  - Add – adds the fragment color to the framebuffer color
  - Blend – combine the fragment and framebuffer color using blending based on an alpha value
    - Alpha can come from stored pixel information, texture, constant value, or from vertices (each vertex has an alpha value – becomes interpolated)
  - Output framebuffer is used as resource texture for the next pass

Shader Scene Node

- May want ability to change shaders within scene graph
  - Different effects for different objects
  - Rather than one “master” shader program to handle all effects use different shaders
    - More efficient, compact shader code for each
- Possible design
  - Compile and link shader code in constructor
    - Can use ShaderSupport code
  - During Draw, call glUseProgram
  - Reset to prior program (or program 0) after children are drawn
  - Place near the root node as switching programs is expensive
- Need a way to handle different uniform variables within other scene nodes
  - Not all shaders need each other’s uniform variables
  - Carry the set within SceneState and modify only those that are active
    - Use -1 value to indicate inactive

Multitexturing

- **Multitexturing** allows 2 or more textures to be accessed during the same rendering pass
  - Texture blending pipeline
    - Made up of a series of **texture stages**
- Saves rendering passes
- Supported in most graphics accelerators
- Provides support for features such as lightmaps, bump-mapping, gloss mapping, etc.



Simple Example

- http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/multi-texture/
- Example uses same texture coordinates for both textures
  - So no change to vertex shader
  - Fragment shader access both textures:

```
texel = texture(tex,gl_TexCoord[0].st)+
        texture(l3d,gl_TexCoord[0].st);
```

Texture Unit 1

Texture Unit 2