Applying Textures

Lecture 27

Robb T. Koether

Hampden-Sydney College

Fri, Nov 3, 2017

Outline

Outline

Applying Textures

- To apply a texture that was internally generated (e.g., the checkerboard), we must do the following
  - Create a texture "name" – The name is an unsigned integer.
  - Generate a texture - OpenGL will assign a value to the name.
  - "Bind" the texture – Create the (empty) texture object.
  - Copy the texels to the texture object.

Applying Textures

- To apply a texture that was internally generated (e.g., the checkerboard), we must do the following
  - Create a texture "name" – The name is an unsigned integer.
  - Generate a texture - OpenGL will assign a value to the name.
  - "Bind" the texture – Create the (empty) texture object.
  - Copy the texels to the texture object.
- This is the same pattern that we used for vertex buffer objects and vertex array objects.

Applying Textures

Applying Textures

```
GLuint checkerboard;
glGenTextures(1, &checkerboard);
glBindTexture(GL_TEXTURE_2D, checkerboard);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
    8, 8, 0, GL_RGB, GL_UNSIGNED_BYTE,
    tex_checkerboard_data);
```

Outline

Photographs as Textures

- The file `vdds.cpp` contains functions that *greatly* simplify converting photographs to textures.
- The functions `vglLoadImage()` and `vglLoadTexture()` do all the work for us.
- `vglLoadTexture()` creates internally an `vglImageData` object.
- After it has copied the texture to the shaders, it deletes the `vglImage` Data object.

The `vglImageData` Object

The `vglImageData` Object

```
struct vglImageData
{
    GLenum target;              // E.g., GL_TEXTURE_2D
    GLenum internalFormat       // E.g., GL_RGB
    GLenum format               // E.g., GL_RGB
    GLenum type;                // E.g., GL_RGB
    GLsizei mipLevels;          // Number of mipmap levels
    GLsizei slices;             // For arrays
    GLsizeiptr sliceStride;     // Distance between slices
    GLsizeiptr totalDataSize;   // Total number of bytes
    vglImageMipData mip[];      // Mipmap data
};
```

The `vglLoadImage()` Function

The `vglLoadImage()` Function

```
void vglLoadImage(const char* filename,
    vglImageData* image);
```

- The `vglLoadImage()` function reads the image data from the specified file and stores it in a `vglImageData` object.
- The file type must be `.dds` (DirectDraw Surface), a Microsoft creation.

The `vglLoadImage()` Function

The `vglLoadImage()` Function

```
void vglLoadImage(const char* filename,
    vglImageData* image);
```

- The `vglLoadImage()` function reads the image data from the specified file and stores it in a `vglImageData` object.
- The file type must be `.dds` (DirectDraw Surface), a Microsoft creation.
- We do not need to call this function directly, because it is called by the next function...

The `vglLoadTexture()` Function

The `vglLoadTexture()` Function

```
void vglLoadTexture(const char* filename,
    GLuint texture,
    vglImageDAta* image);
```

- The `vglLoadTexture()` function
  - Creates a `vglImageData` object.
  - Reads the image into it.
  - Generates and binds the texture object.
  - Copies the data from the `vglImageData` object to the texture object.
  - Destroys the `vglImageData` object.

Outline

The Application Program

The Application Program

```
enum {vPosition = 0, ..., vTexture = 3};
enum {GrassTex, RoadTex, ..., NumTextures};
GLuint Texture[NumTextures];
glGenTextures(NumTextures, Texture);
```

- We follow the same pattern used for VBOs and VAOs.

The Application Program

The Application Program

```
vglLoadTexture("pathname/grass.dds", Texture[GrassTex],
    &image);
vglLoadTexture("pathname/road.dds", Texture[RoadTex],
    &image);
        ⋮
```

- We follow the same pattern used for VBOs and VAOs.

Outline

The Vertex Array Object

The Vertex Array Object

```
struct VertexDataTex3D
{
    vec3 pos;
    vec3 norm;
    vec2 tex;
};
```

- Just as we added the coordinates of the normal vectors for each vertex, we must add the texture coordinates for each vertex.

The Vertex Array Object

The Vertex Array Object

```
glVertexAttribPointer(vTexture, 2, GL_FLOAT, ...);
glEnableVertexAttribArray(vTexture);
```

- Describe the structure of the buffer, including the texture coordinates.

Outline

The Vertex Shader

The Vertex Shader

```
layout (location = 3) in vec2 vTexture;
out vec2 tex_coord;
tex_coord = vTexture;
```

- In the vertex shader, we need only the texture coordinates for the vertex.
- They should be included in the vertex buffer along with the position and normal vectors.
- These coordinates will be passed to the fragment shader, where they will be interpolated across the primitive.

Outline

The Fragment Shader

The Fragment Shader

```
uniform sampler2D tex;
in vec2 tex_coord;      // From the vertex shader
```

- A **`sampler2D`** object is the uniform variable that holds the texture object.

The `texture()` Function

The `texture()` Function

```
vec4 texel = texture(tex, tex_coord);
```

- The `texture()` function uses the texture coordinates to return a **`vec4`** object (RGBA) from the array of texels.
- This value is used in place of the color (ambient and diffuse) or is blended with the color.

The Fragment Shader

```
uniform bool use_tex;
```

- We could add a **`uniform bool`**, which can be set independently for each object, that is `true` if textures are to be applied and `false` otherwise.
- If `use_tex` is true, then `texel` is used.
- If `use_tex` is false, then the material colors are used.

Outline

Homework

Homework

- Read pages 259 - 263: Texture Mapping & Basic Texture Types
- Read pages 270 - 277: Texture Formats