Texture Coordinates

Dinesh K. Pai

Textbook Chapter 15

Some slides courtesy of M. Kim, KAIST

Today

- Reminders:
  - Assignment 3 due today
  - Midterm 2 coming soon (March 21)
- Assignment 4 introduction
- Cube maps
- Projector maps
Assignment 4 demo

C³ Survey

How far along are you with Assignment 3
a) Not started
b) Can run template code
c) Finished at least one part
d) Finished all fully specified parts (1,2,3)
e) Finished everything
C³ Survey

- Assignment 4 will be out soon (tomorrow). It is very useful for understanding texture mapping and studying for midterm 2 on March 21. When should we make Assignment 4 due?
  a) March 19 (2 working days before midterm)
     + makes sure you finish assignment before midterm
     - many of you have lots going on in the next 2 wks
  b) March 25 (2 working days after midterm)
     - some may procrastinate on assignment and hence lose learning opportunity for midterm
     + gives you more time, but … CAVEAT: don't complain that you didn’t do the assignment later! Please do at least parts 1-3 before midterm

C³ Review: Texture mapping

- In which part of the pipeline can you access textures?
  a) In the vertex shader
  b) In the fragment shader
  c) Both of the above
  d) None of the above
C³ Exercise: Texture mapping

- If the following picture corresponds to the texture coordinates:

```
static GLfloat sqTex[12] = {
  0, 0,
  1, 1,
  1, 0,
  0, 1, 
  1, 1
};
```

which picture corresponds to the following?

```
static GLfloat sqTex[12] = {
  0, 0,
  1, 1,
  1, 0,
  0, 1, 
  0, 0,
  1, 0,
  1, 1
};
```

![Image](image.png)

(a)  (b)  (c)  (d)  None of these. It's an error.

More on Texture Coordinates

- Part 1 uses the texture coordinates supplied with the model, generated using a 3rd party program (3DS Max). Similar functions available in Blender, Maya, and other modeling software.
- Legacy OpenGL had a function (glTexGen) to do this, removed from current versions.
- In production, coordinates designed with model (or “painted” on 3D model).
- The next two parts show how useful texture coordinates can often be computed in shaders.
Environment cube maps

- Textures can also be used to model the environment in the distance around the object being rendered.
- In this case, we typically use 6 square textures representing the faces of a large cube surrounding the scene.

Environment cube maps

- Each texture pixel represents the color as seen along one direction in the environment.
- This is called a cube map. GLSL provides a cube-texture data type, `samplerCube` specifically for this purpose.
Environment cube maps

- During the shading of a point, we can treat the material at that point as a perfect mirror and fetch the environment data from the appropriate incoming direction.

Environment map shader

- We calculated $B(\vec{\nu})$ in a previous lecture.
- This bounced vector will point towards the environment direction, which would be observed in a mirrored surface.
- By looking up the cube map, using this direction, we give the surface the appearance of a mirror.
Geometry of Cube Mapping

Environment map shader

- Fragment shader

```glsl
#version 330
uniform samplerCube uTexUnit0;
in vec3 vNormal;
in vec4 vPosition;
out vec4 fragColor;

vec3 reflect(vec3 w, vec3 n){
    return n*(dot(w,n)*2.0) - w; // bounce vector
}

void main() {
    vec3 normal = normalize(vNormal);
    vec3 reflected = reflect(normalize(vec3(-vPosition)), normal);
    vec4 texColor0 = textureCube(uTexUnit0, reflected);
    fragColor = vec4(texColor0.r, texColor0.g, texColor0.b, 1.0);
}
```

From book:

Can also use built-in GLSL

Note: \( B(\vec{w}) = \text{reflect}(\vec{w}) \)
### Environment map shader

- \( \mathbf{v_{Position}} \) represents the view vector

- `textureCube` is a special GLSL function that takes a direction vector and returns the color stored at this direction in the cube texture map.

- Here we assume eye-coordinates, but frame changes may be needed.

### Projector texture mapping

- There are times when we wish to glue our texture onto our triangles using a *projector* model, instead of the affine gluing model.

- For example, we may wish to simulate a slide projector illuminating some triangles in space.
Geometry of Projector Textures

Projector texture mapping

- The slide projector is modeled using 4 by 4, modelview and projection matrices, $M_s$ and $P_s$

\[
\begin{bmatrix}
    x_w \\
    y_w \\
    -w \\
    w
\end{bmatrix}
= P_s M_s
\begin{bmatrix}
    x_o \\
    y_o \\
    z_o \\
    1
\end{bmatrix}
\]
Projector texture mapping

- With the texture coordinates defined as
  \[ x_t = \frac{x_i w_t}{w_i} \quad \text{and} \quad y_t = \frac{y_i w_t}{w_i} \]

- To color a point on a triangle with object coordinates \([x_o, y_o, z_o, 1]^T\), we fetch the texture data stored at location \([x_t, y_t]^T\)

- The three quantities \(x_i w_t, y_i w_t\) and \(w_t\) are all affine functions of \((x_o, y_o, z_o)^T\). Thus these quantities will be properly interpolated over a triangle when implemented as varying variables.

- In the fragment shader, we need to divide by \(w_t\) to obtain the actual texture coordinates.

- When doing projector texture mapping, we do not need to pass any texture coordinates as attribute variables to our vertex shader.
**Projector texture mapping**

- We simply use the object coordinates already available to us.
- We do need to pass in, using uniform variables, the necessary projector matrices.

---

**Projector vertex shader**

```glsl
#version 330

uniform mat4 uModelViewMatrix;
uniform mat4 uProjMatrix;

uniform mat4 uSProjMatrix;
uniform mat4 uSModelViewMatrix;

in vec4 aVertex;
out vec4 vTexCoord;

void main(){
    vTexCoord = uSProjMatrix * uSModelViewMatrix * aVertex;
    gl_Position = uProjMatrix * uModelViewMatrix * aVertex;
}
```

Vertex shader generates texture coordinates! But not normalized.
Projector texture mapping

- Projector fragment shader

```glsl
#version 330

uniform sampler2D vTexUnit0;

in vec4 aTexCoord;
out vec4 fragColor;

void main(){
  vec2 tex2;
  tex2.x = aTexCoord.x/aTexCoord.w;
  tex2.y = aTexCoord.y/aTexCoord.w;
  vec4 texColor0 = texture2D(vTexUnit0, tex2);
  fragColor = texColor0;
}
```

Conveniently, OpenGL even gives us a special call `texture2DProj(vTexUnit0, pTexCoord)`, that actually does the divide for us.

Inconveniently, when designing our slide projector matrix `uSProjMatrix`, we have to deal with the fact that the canonical texture image domain in OpenGL is the unit square, whose lower left and upper right corners have coordinates `[0,0]` and `[1,1]` used for the display window.