Texture Coordinates

Dinesh K. Pai

Textbook Chapter 13,15

Some slides courtesy of M. Kim, KAIST

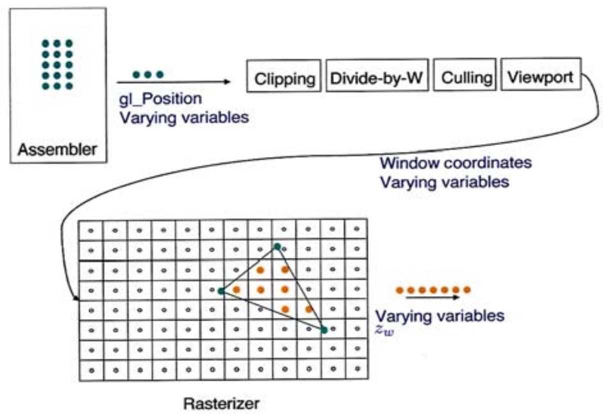
1

Today

- Announcements
  - Assignment 4 will be out soon. Due April 2.
  - Reminder: Quiz 3 will be on March 27
  - I have to attend a meeting downtown on Thursday morning. My office hour (10-11) will be covered by TA (Joao) in x432. Knock on the door.
- Texture mapping in WebGL
- Review of projection
- Perspective-correct interpolation
- Projector maps

2

Path from vertex to pixel



3

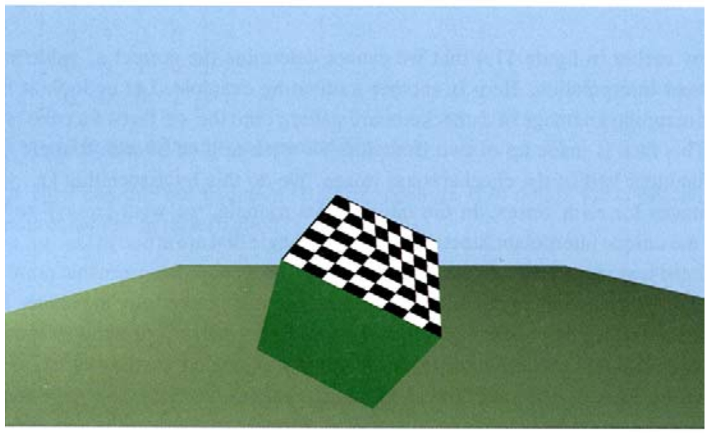
Interpolation of varying variables

- Topic of Chapter 13. Optional for this course, but please remember that there is a subtle issue.
- In between the vertex and fragment shader, we need to interpolate the values of the varying variables.
- This is surprisingly subtle (called “perspective correct interpolation”).

4

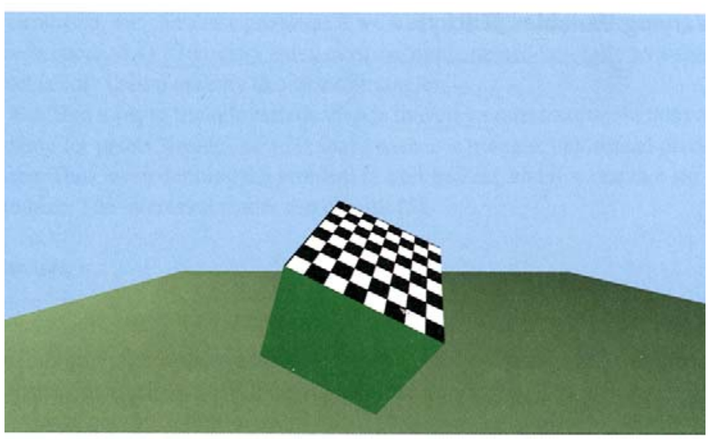
Wrong representation of texture

When texture coordinates are linearly interpolated in window coordinates, an incorrect image results.



5

Correct representation of texture



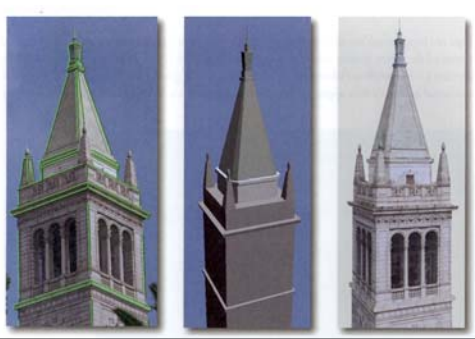
6

- Recap of Projection (see Last part of L15)

7

Projector texture mapping

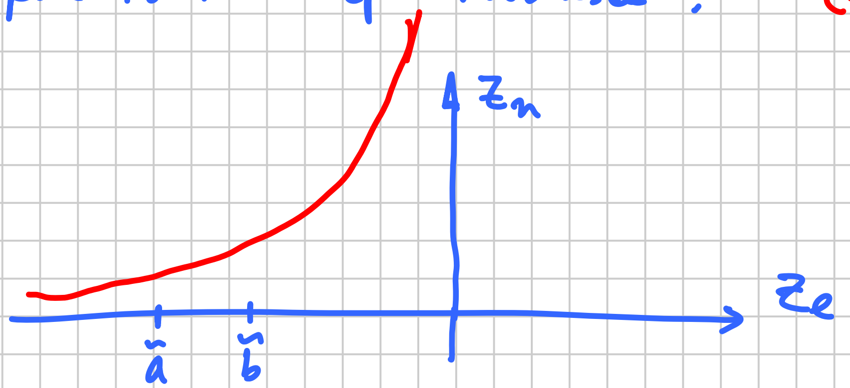
- There are times when we wish to glue our texture onto our triangles using a *projector* model, instead of the affine gluing model.
- For example, we may wish to simulate a slide projector illuminating some triangles in space.



8

Depth & projection

What happens to the depth variable? (see Sec. 11.2 of Text)



Observe that this is a monotonic function.

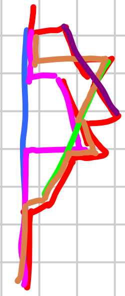
$\breve{a}$ is farther than $\breve{b}$, if. and only if

$$z_n(\breve{a}) < z_n(\breve{b})$$

Store the depth value z_n of the nearest fragment as a number at each pixel location in an "image" called the **Z-buffer** or **depth buffer.**

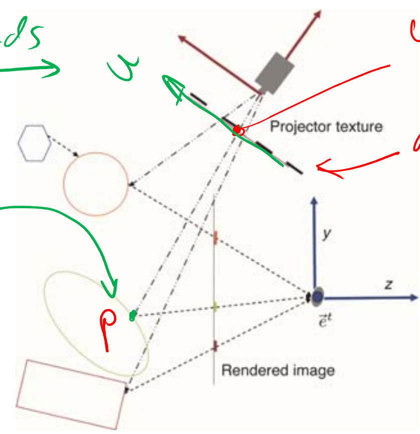
Looks like a grey scale image, brighter ⇒ closer

Z-buffer



— depth buffer after green line

Geometry of Projector Textures



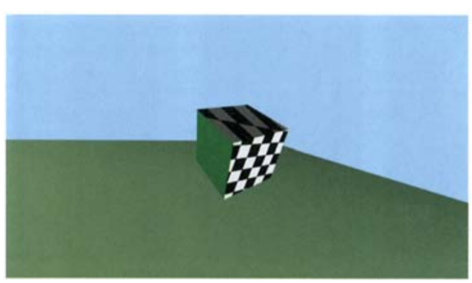
9

Projector texture mapping

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

you should compute in JavaScript, pass as uniform

$$\begin{bmatrix} x_t w_t \\ y_t w_t \\ - \\ w_t \end{bmatrix} = P_s M_s \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$



10

Projector texture mapping

- With the texture coordinates defined as

$$x_t = \frac{x_t w_t}{w_t} \text{ and } y_t = \frac{y_t w_t}{w_t}$$

- 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 $[x_t, y_t]^t$

11

Projector texture mapping

- The three quantities $x_t w_t$, $y_t w_t$ and w_t are all affine functions of (x_o, y_o, z_o). 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.

12

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 texture mapping

- Projector vertex shader

```
#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

```
#version 330

uniform sampler2D vTexUnit0;

in vec4 aTexCoord;
out vec4 fragColor;

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

varying

15

Projector texture mapping

- 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 $[0,0]^t$ and $[1,1]^t$ used for the display window.

16