Tutorial 7

CS3241 Computer Graphics (AY23/24)

October 24, 2023

Wong Pei Xian

e0389023@u.nus.edu

Recap of Texture Mapping

Textures

- Arguably the single most important feature of any graphics engine is to support textures
  - Object Texture mapping
  - Environment mapping
  - Antialiasing (mipmaps)
  - Texture objects/Framebuffer objects

Local parameters (Bound texture)

```
void init( void ) {
  ...
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

  glGenTextures(1, &texObj);

  glBindTexture(GL_TEXTURE_2D, texObj);

  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, imageWidth, imageHeight,
               0, GL_RGBA, GL_UNSIGNED_BYTE, texImage);
  ...
}
```

Local to the currently bound texture.

Global state (Texture Environment)

```
void display( void )  {
  ...
  glEnable(GL_TEXTURE_2D);

  glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);

  glBindTexture(GL_TEXTURE_2D, texObj);

  glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-2.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(0.0, -1.0, 0.0);
    glTexCoord2f(0.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(2.4, 1.0, -1.4);
    glTexCoord2f(1.0, 0.0); glVertex3f(2.4, -1.0, -1.4);
  glEnd();
  ...
}
```

Global state (which env and env mode)

Texture Environments

Texture Functions / Environments

- Texture function can be set using

  `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, f );`

  where `f` is `GL_DECAL`, `GL_REPLACE`, `GL_MODULATE`, `GL_BLEND`, `GL_ADD`, or `GL_COMBINE`

- Defines how the texture color is combined with the underlying primary color of the fragment

Color

Alpha

Texture Base Internal Format	Texture source color C_s	A_s
ALPHA	$(0, 0, 0)$	A_t
LUMINANCE	(L_t, L_t, L_t)	1
LUMINANCE_ALPHA	(L_t, L_t, L_t)	A_t
INTENSITY	(I_t, I_t, I_t)	I_t
RGB	(R_t, G_t, B_t)	1
RGBA	(R_t, G_t, B_t)	A_t

Texture function

A **texture function** acts on the fragment to be textured using the texture image value that applies to the fragment and produces an RGBA color for that fragment.

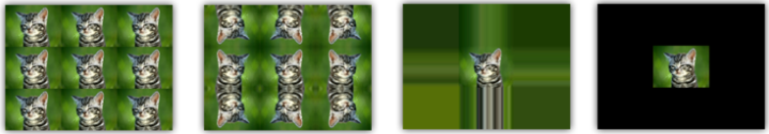
- C = color (RGB) $\in [0, 1]$
- A = alpha (A) $\in [0, 1]$
- $_p$ = color from previous texture stage/incoming fragment
- $_s$ = texture source color
- $_c$ = texture environment color
- $_v$ = final value produced

Common texture functions

Texture Base Internal Format	Value	GL_REPLACE Function	GL_MODULATE Function
GL_ALPHA	$C_v =$	C_p	C_p
	$A_v =$	A_s	$A_p A_s$
GL_LUMINANCE	$C_v =$	C_s	$C_p C_s$
(or 1)	$A_v =$	A_p	A_p
GL_LUMINANCE_ALPHA	$C_v =$	C_s	$C_p C_s$
(or 2)	$A_v =$	A_s	$A_p A_s$
GL_RGB	$C_v =$	C_s	$C_p C_s$
(or 3)	$A_v =$	A_p	A_p
GL_RGBA	$C_v =$	C_s	$C_p C_s$
(or 4)	$A_v =$	A_s	$A_p A_s$

Texture wrapping modes

GL_REPEAT GL_MIRRORED_REPEAT GL_CLAMP_TO_EDGE GL_CLAMP_TO_BORDER

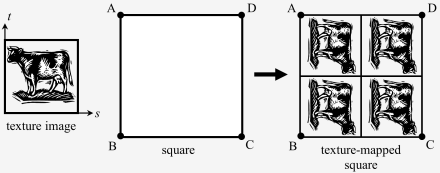


GL_REPEAT GL_MIRRORED_REPEAT GL_CLAMP_TO_EDGE GL_CLAMP_TO_BORDER

Questions

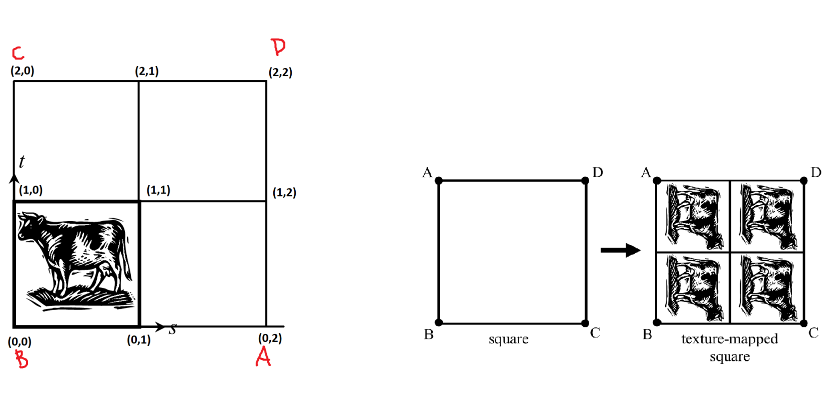
Question 1

Suppose the texture coordinate wrapping mode has been set to `GL_REPEAT` for both the *s* and *t* texture coordinates.



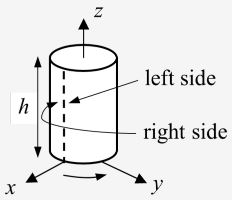
Given a texture image and a square as shown in the following diagram, what are the 2D texture coordinates assigned to vertices A, B, C and D so that texture-mapped square appears as shown below?

Question 1


C
D
(2,0)
(2,1)
(2,2)
t
(1,0)
(1,1)
(1,2)
(0,0)
(0,1)
s
(0,2)
B
A
A
D
B
C
square
A
D
B
C
texture-mapped
square

Question 2

We would like to wrap an entire texture map around the side of the cylinder shown in the diagram.



The center of the base of the cylinder is located at (0, 0, 0). Given any point (p_x, p_y, p_z) on the side of the cylinder, write the two expressions to compute the s and t texture coordinates at the point.

Question 2

$s =$ arc length between vector (1,0) and $(p_x, p_y)\ /2\pi$
$t = p_z/h$

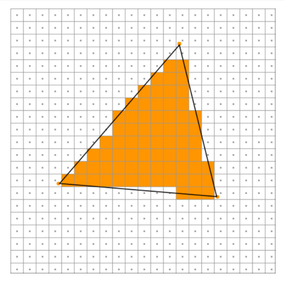
$s \in [0, 1] \mapsto [0, 2\pi]$
$t \in [0, 1] \mapsto [0, h]$

An aside into **Aliasing**

Nyquist rate: $2\times$ the highest frequency of a function/signal

Aliasing occurs when **sampling at a rate below** the Nyquist rate.

What's being sampled here?



If we only sample one point per texel, we are definitely sampling below the Nyquist rate (which would be 2 points per pixel).
Effects appear as **jaggies** or **Moire** patterns.

Examples of moire

Also see video example.

3 contributors to aliasing

The aliasing problem occurs:

1. Along edges primitive's edges
2. In tiny objects
3. Within complicated detail

Solutions to aliasing

1. Increase resolution
2. Low-pass filter
   - Contour smoothing/box filter areas on screenspace to sample
3. Prefiltering
   - Prefilters of different levels as **mipmaps**
   - Makes each fragment represent **the average** of a finite area in scene
   - As opposed to a single point
4. Increase resolution

Most of **antialiasing** today is based on a combination of the 3.

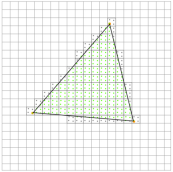
Types of AA

- FSAA/SSAA: Fullscreen/Super sampling anti-aliasing
- MSAA: Multisample anti-aliasing
- FXAA (Lottes sampling): Fast approximate anti-aliasing
- SMAA: Enhanced Subpixel Morphological Anti-aliasing (combined modular method)
- TAA: Temporal Anti-aliasing (sampling frames)

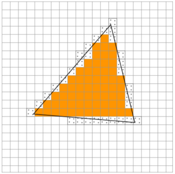
Excellent summary paper by Kesten (2017)

Overview of SSAA, MSAA, FXAA

Uses point sampling of unrasterized polys

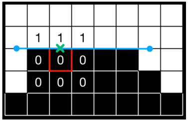


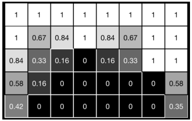
SSAA [x4]



MSAA [x4] = SSAA + Edge detection

Uses a deterministic algorithm based on neighbouring fragment luma values

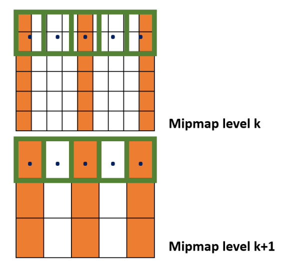




FXAA = Fragment processing

Why mipmaps then?

By prefiltering the higher resolution (aka higher frequency with higher Nyquist rate) maps, we get lower resolution maps. The sampled values thus change less often.

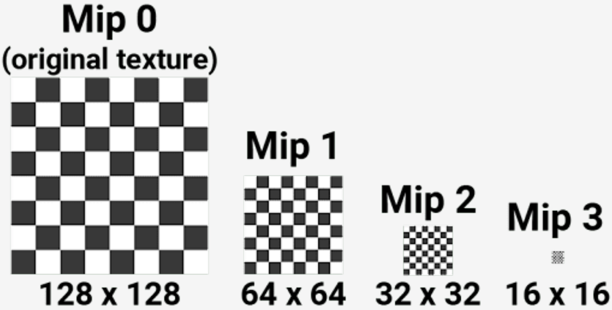


Question 3a

Given a 512 $\times$ 512 texture image, we want to create a mipmap from it and use the appropriate mipmap level during rendering.

How many levels are in the mipmap? (including the original texture level)

Question 3a



Mipmap level 0 is original texture. Additionally can halve it's length and width $\log(n)$ times (given $n = \min(h, w)$).

$$\log(512) + 1 = 10$$

Question 3b

The mipmap is used to texture-map a 3D square that appears in a 100 × 100 region on the screen. What is the best integer mipmap level to use to texture-map the square? Assume that we *prefer a more blurred rendered image* if the exact level is not an integer.

The highest-resolution texture image is level 0, the next is level 1, and so on.

Question 3b

Manually halving the dimension: $512 \rightarrow 256 \rightarrow 128 \rightarrow 64 < 100$

$$\lceil \log(512/100) \rceil = \lceil \log(5.12) \rceil = 3$$

Question 3c

Some rendering systems can actually take non-integer mipmap level and compute the result by interpolating between two mipmap levels. (aka ***Trilinear interpolation***)

In the case of Part (b), what is the exact mipmap level to use to texture-map the square? Round your answer to 2 decimal places. You can use the formula $\log_2(x) = \log_{10}(x)/\log_{10}(2)$.

Trilinear interpolation

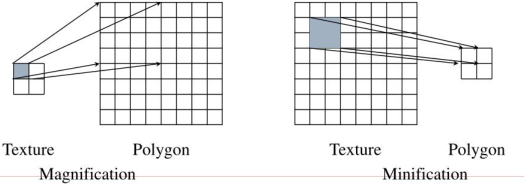
1. Mip level 2 (128×128): Sample value A
   - $\log(128) = 7$
2. Mip level 3 (64×64): Sample value B
   - $\log(64) = 6$
3. $\log(100) \approx 6.643$
4. Thus the sampled value hsould be interpolated between A and B as `lerp(B, A, fract(log(100)))` where `fract` returns the fractional component of the number returned.

Magnification and Minification

Magnification and Minification

More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

Can use point sampling (nearest texel) or linear filtering (2 x 2 filter) to obtain texture values



Question 4

Given a gray-scale image $I[1 \ldots W, 1 \ldots H]$, we can compute a special table $S[1 \ldots W, 1 \ldots H]$ such that $S[m, n] = \sum_{i=1}^{m} \sum_{j=1}^{n} I[i \ldots j]$. Then using S, the sum of pixel values within any rectangular region of the range $I[x_i \ldots x_2, y_1 \ldots y_2]$ can be computed in constant time:

$$S[x_2, y_2] - S[x_1 - 1, y_2] - S[x_2, y_1 - 1] + S[x_1 - 1, y_1 - 1]$$

Summed Area Table

Summed Area Table

Regular image

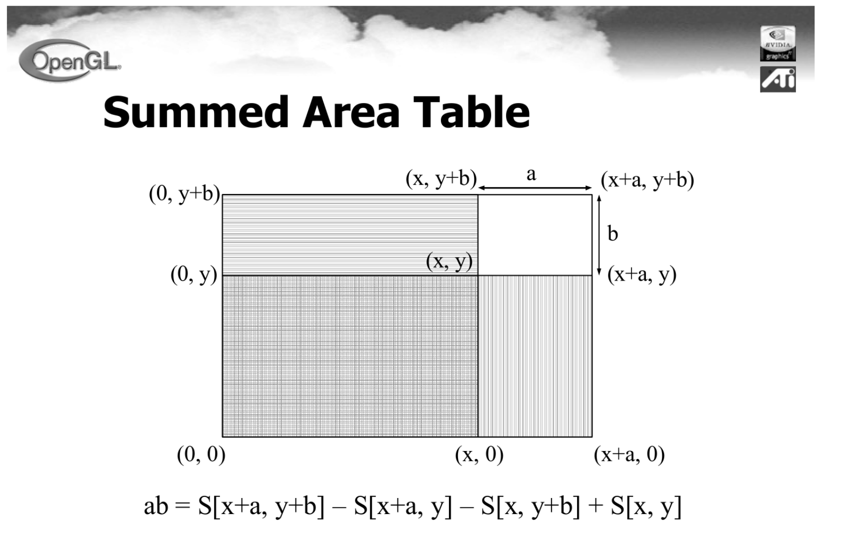
1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

→

Summed area table

4	8	12	16
3	6	9	12
2	4	6	8
1	2	3	4

Summed Area Table



Question 4a

What could be the use of the special table S for overcoming a particular problem in texture mapping? Explain why it is suitable.

Solves **aliasing**!

We can approximate the pre-image of a fragment as a rectangle, and use the special table S to quickly find the average of the texel values within the rectangular region.

- efficient area sampling within **any rectangular region**
- S can be precomputed before real-time rendering

Question 4b

Compare the quality of the rendered image produced by the technique you have learned in class, with that using the special table S? What attributes to the difference?

Quality comparison

The rendered image will look better and sharper with Summed Area Tables.

Textures with SAT are a **more accurate approximation** of the pre-image region of the fragment (**arbitrary region in constant time**).

Question 5a

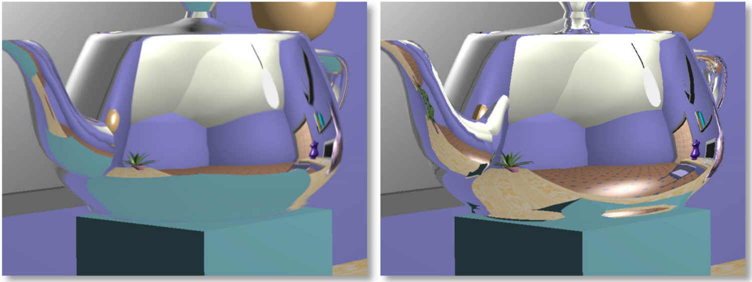
A reflective object can be rendered using reflection mapping or ray tracing. List two situations where there will be obvious differences between the images produced by the two methods.

Situation 1: Self-intersection

The object itself is not part of the environment!

Situation 2: Relatively large object

When the reflective object is quite large compared to the size of its surrounding.



environment mapping ray tracing

Question 5b

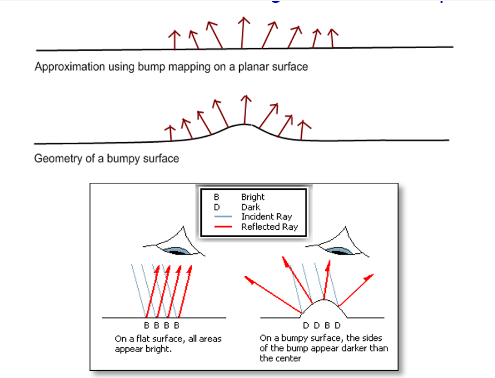
Describe a way you can use to detect that the features on an object are actually bump-mapped instead of real geometry.

Question 5c

Suppose you can extend the functionalities in any stage of the raster graphics pipeline. We want to render polygons with bump-mapping.

1. Should the lighting computation be performed per fragment, per vertex, or per polygon?
2. At which pipeline stage should the lighting computation be performed?

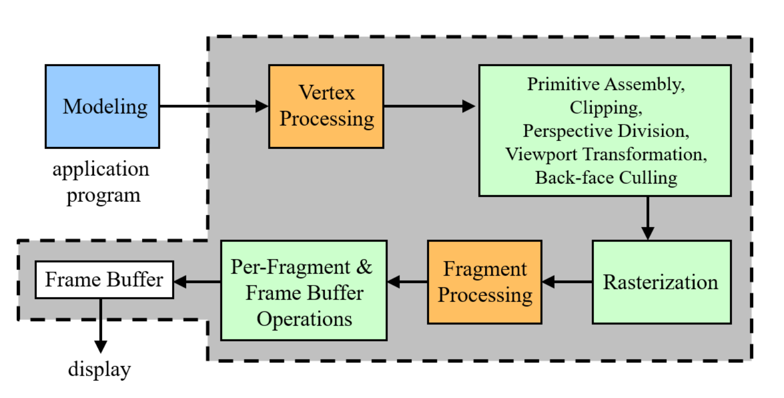
Bump mapping



Normals provided by the bump map **per fragment** to do lighting computation with (e.g. Phong Illumination Equation).

Lighting computation

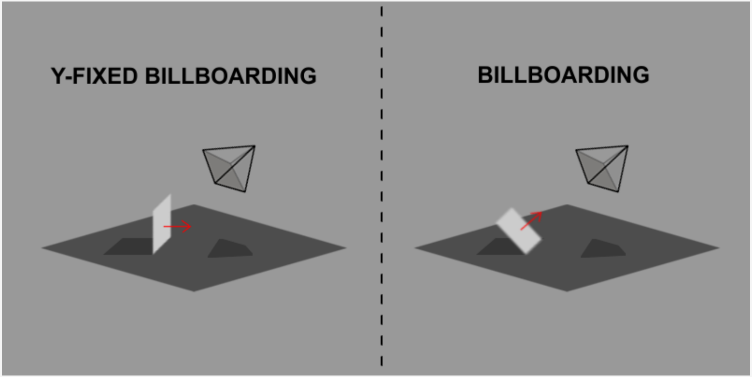
Probably in the fragment processing stage



Question 5d

In a virtual outdoor scene, each tree is rendered using a vertical rectangular billboard. The vertical axis of the world is the z-axis. The viewpoint is at (v_x, v_y, v_z). For a billboard whose center is at (b_x, b_y, b_z), what should be the normal vector of the billboard rectangle? You need not normalize the vector.

Question 5d



- z-axis is fixed: $(v_x - b_x, v_y - b_y, 0)$
- completely billboarded: $(v_x - b_x, v_y - b_y, v_z - b_z)$

Attendance taking

Thanks! Get the slides here after the tutorial.

https://trxe.github.io/cs3241-notes