UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4370 Fall 2023

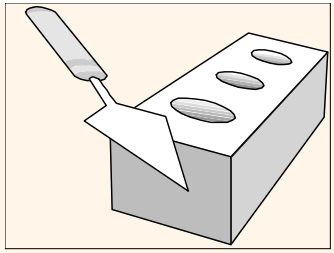
Interactive Computer Graphics

M & W 5:30 to 7:00 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4370

5:30 to 7

PLEASE

LOG IN

CANVAS

Please close all other windows.

NEXT.

10.18.2023 (W 5:30 to 7) (17)	Homework 7	**Lecture 8** **(Discrete Techniques)**
10.23.2023 (M 5:30 to 7) (18)		**Lecture 9** **(Programmable Shaders)**
10.25.2023 (W 5:30 to 7) (19)		**Lecture 10** **(Modeling and Hierarchy)**
10.30.2023 (M 5:30 to 7) (20)		**PROJECT 3**
11.01.2023 (W 5:30 to 7) (21)		**EXAM 3 REVIEW**
11.06.2023 (M 5:30 to 7) (22)		**EXAM 3**

PROJECTS - 30% — 30% of Total

PROJECT 3
PROJECTS 30% Module | **Not available until** Oct 16 at 7:00pm | **Due** Oct 30 at 5:30pm | 100 pts

VH, publish

PROJECT 3

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. The "*PROJECT 3 Acceptance Testing*" (-20 points)
4. The "*PROJECT 3 Acceptance Testing Check Sheet*": (-10 points)
   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)
   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT
5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

For the Grader: if the student did not submit, please skip and do not assign a ZERO (MISSING).

Points	100
Submitting	a file upload

Due	For	Available from	Until
Oct 30 at 5:30pm	Everyone	Oct 16 at 7pm	Oct 30 at 5:30pm

PROJECT 3

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. *The "PROJECT 3 Acceptance Testing ".* (-20 points)
4. *The "PROJECT 3 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 3
(100 points)

Hours spent:

The PROJECT is to be turned in before 5:30 PM of the due date class.

Submit:

1. ***The zipped Visual Studio2019 or WEBGL "PROJECT 3".***
2. ***The "PROJECT 3 Acceptance Testing".***
3. ***The "PROJECT 3 Acceptance Testing Check Sheet" as "score.OPENGL.doc or score.WEBGL.doc" to CANVAS.***
4. ***The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".***

S'COOL BUS - FALL 2023 Color & Shading and Texturing Instructions
(The file "**S'COOL BUS FALL 2023**" contains the specifications for the BUS that you started modeling.)

1. The **glutCreateWindow** OpenGL call will take as parameter "**YourLastName** FirstName **BUS Version 3**".

2. Start with the **MyBus.c** that you have created in Programming Assignment 2 and call the file **LastNameBus.c.**

3. Lighting and Shading:
Add 1 **spot light** initially located on the **top inside right** of (right side looking at the BUS looking from the FRONT) the BUS, distance term a = 1, initial direction towards the center of the BUS. You can use the coefficients from the Lecture 6 and the format from Homework 6 for the MaterialStruct.
You should allow the movement of the light source in 3 positions on each x and z axis, i.e., **front left**, **middle**, **right** and **middle side left**, **middle**, and **right**.
You should allow three changes of the **angle of the light**, **top**, **middle**, and **bottom**.

4. Texturing:
Texture your **BUS** by using either a texture similar to the actual **BUS** or any other BUS siding textures or bump mapping.
If you have several texturing, create a menu to use all of them. I guess this could be the extra credit problem. (20 points)

5. Navigating:
First, the **discrete reposition & orientation of the light** should be linked to a menu, key, etc.

Second, the **LookAt** needs to be amended with the following camera positions:

- at the **DOOR** looking toward the middle back BUS, then back **top inside right**, and then front **top inside right**.
- in the **middle of the BUS** looking at the same as above PLUS back towards the **DOOR**.

Attach these fixed positions to a menu, key, etc.

Get screenshots of the "YourLastName FirstName BUS Version 3" for each view and please comment on what the screenshot is about.

PROJECT 3

Publish | Edit

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. *The "PROJECT 3 Acceptance Testing "* (-20 points)
4. *The "PROJECT 3 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 3

Publish | Edit

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. *The "PROJECT 3 Acceptance Testing ".* (-20 points)
4. *The "PROJECT 3 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 3 Acceptance Testing:

Insert screenshots of the output produced and insert it in this file …….

Lighting and Shading:

1. **Place spot light at top inside right** initial direction towards the center of the BUS.
SCREENSHOT Output of your program is:
……..

2. Move light at **front left**.
SCREENSHOT Output of your program is:
……..

3. Move light at **front middle**.
SCREENSHOT Output of your program is:
……..

4. Move light at **front right**.
SCREENSHOT Output of your program is:
……..

5. Move light at **middle side left**.
SCREENSHOT Output of your program is:
……..

6. Move light at **middle side middle**.
SCREENSHOT Output of your program is:
……..

7. Move light at **middle side bottom**.
SCREENSHOT Output of your program is:
……..

……..

Texturing:

1. Texture your **BUS** by using either a texture similar to the actual **BUS** or any other BUS siding textures or bump mapping.
SCREENSHOT Output of your program is:
……..

Words on the BUS:

1. **SCREENSHOT Output of your program is:**
……..

Navigation:

1. **Discrete reposition & orientation of the light** should be linked to a menu, key.
SCREENSHOT Output of your program is:
……..

2. **Second**, the ~~LookAt~~ needs to be amended with the following camera positions: at the **DOOR** looking toward the middle back BUS
SCREENSHOT Output of your program is:
……..

Copy and paste ~~LookAt~~ from your code for this position here:
……..

3. **Second**, the ~~LookAt~~ needs to be amended with the following camera positions: in the **middle of the BUS** looking at the **DOOR**
SCREENSHOT Output of your program is:
……..

Copy and paste ~~LookAt~~ from your code for this position here:
……..

Copy and Paste your .c (.cpp) file here:
……..

PROJECT 3

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. *The "PROJECT 3 Acceptance Testing ".* (-20 points)
4. *The "PROJECT 3 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 3 Hours:

Acceptance Testing Check Sheet:

CHECKLIST (YOU MUST SCORE YOURSELF!):

1. Submitted PROJECT3.zip? 10 points
2. Submitted Acceptance Testing.docx? 10 points
3. Submitted Acceptance Testing Check Sheet as score.OPENGL.docx or score.WEBGL.docx? 10 points
4. Submitted PROJECT 3 PRESENTATION.pptx? 10 points

Lighting and Shading:

1. ACCEPTANCE TESTING Step 1? 5 points
2. ACCEPTANCE TESTING Step 2? 5 points
3. ACCEPTANCE TESTING Step 3? 5 points
4. ACCEPTANCE TESTING Step 4? 5 points
5. ACCEPTANCE TESTING Step 5? 5 points
6. ACCEPTANCE TESTING Step 6? 5 points
7. ACCEPTANCE TESTING Step 7? 5 points

Texturing:

1. ACCEPTANCE TESTING Step 1? 5 points

Text on the BUS:

1. ACCEPTANCE TESTING Step 1? 5 points

Navigation:

1. ACCEPTANCE TESTING Step 1? 5 points
2. ACCEPTANCE TESTING Step 2? 5 points
3. ACCEPTANCE TESTING Step 3? 5 points

Score

I CERTIFY THAT THIS IS MY OWN WORK and THE CHECKBOXES reflect the completion of these DELIVERABLES.

PROJECT 3

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 3.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 3".* (-30 points)
3. The "*PROJECT 3 Acceptance Testing*". (-20 points)
4. The "*PROJECT 3 Acceptance Testing Check Sheet*". (-10 points)

   Rename it to score.OpenGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

5. *The ppt "PROJECT 3 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 3 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

COSC 4370 – Computer Graphics

Lecture 8

Discrete Techniques

Chapter 8

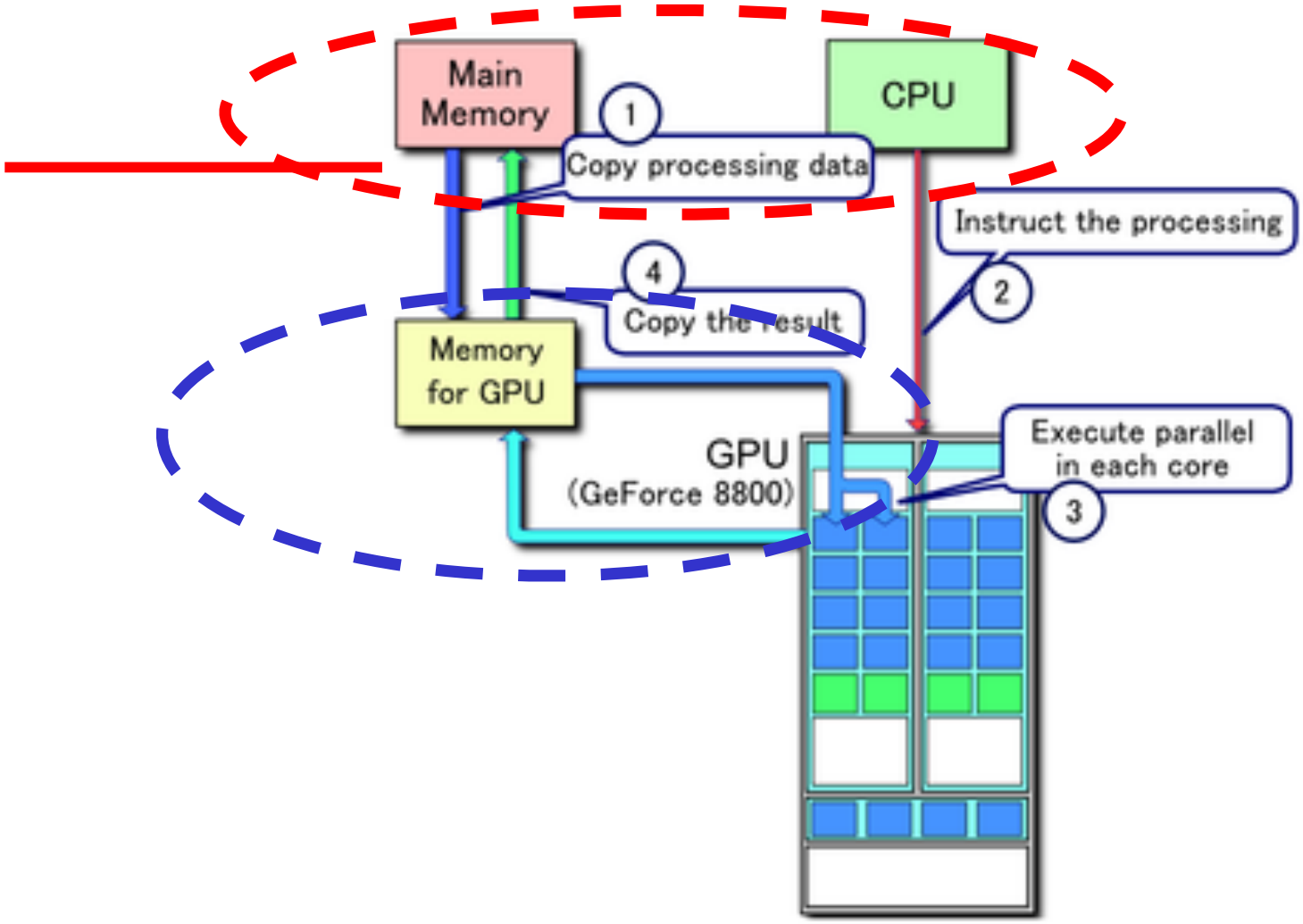
Discrete Techniques

For many years, computer graphics dealt exclusively with geometric objects, such as lines, polygons, and polyhedra. Raster systems have been in existence for more than 30 years, but until recently, **application programmers have had only indirect access to pixels in the frame buffer**. Although lines and polygons were rasterized into the **frame buffer**, application programmers had no functions in the **API** that would allow them to **read or write individual pixels**. Many of the most exciting methods that have evolved over the past decade rely on interactions among the various **buffers** that make up the **frame buffer**. **Texture mapping, antialiasing, compositing, and alpha blending are only a few of the techniques that become possible when the API allows us to work with discrete buffers.**

We start by looking at the frame buffer in more detail and how we can read and write to it. We will learn to work with **arrays of pixels** that form digital images. We then consider mapping methods. These techniques are applied during the **rendering process**, and they enable us to give the illusion of a surface of great complexity, although the surface might be a single polygon.

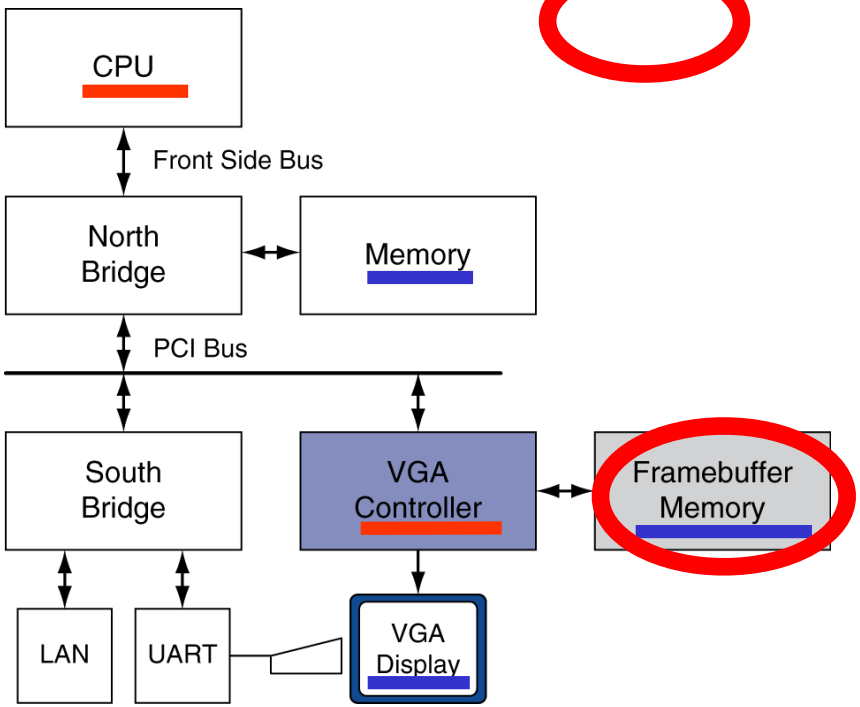
All these techniques use **arrays of pixels** to define how the rendering process that we studied in Chapter 7 is augmented to create these illusions. **We will then look at some of the other buffers that are supported by the OpenGL API and how these buffers can be used for new applications. In particular, we examine techniques for combining or compositing images.** Here we use the **fourth color in RGBA mode**, and we will see that we can use this channel to blend images and to create effects such as **transparency**. We conclude with a discussion of the aliasing problems that arise whenever we work with discrete elements.

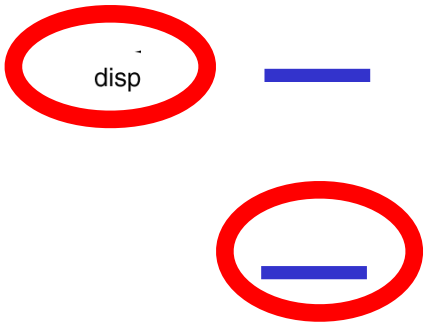
CPU & GPU



1. Copy data from **Main Memory** to **GPU Memory**
2. **CPU** instructs the process to **GPU**
3. **GPU** execute parallel in each core
4. Copy the result from **GPU Memory** to **Main Memory**

Graphics in the System





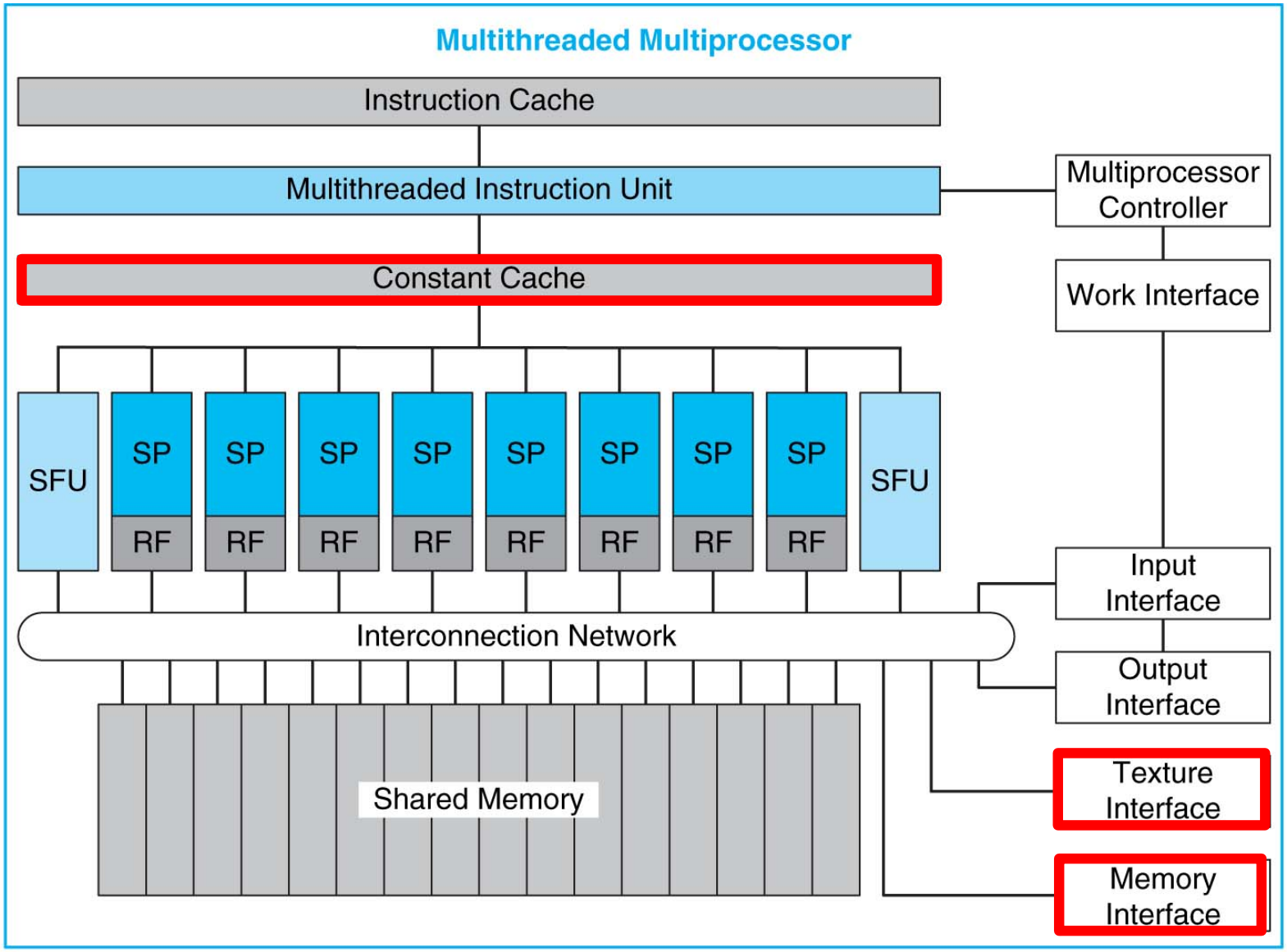


FIGURE A.4.1 Multithreaded multiprocessor with eight scalar processor (SP) cores. The eight SP cores each have a large multithreaded register file (RF) and share an instruction cache, multithreaded instruction issue unit, constant cache, two special function units (SFUs), interconnection network, and a multibank shared memory.

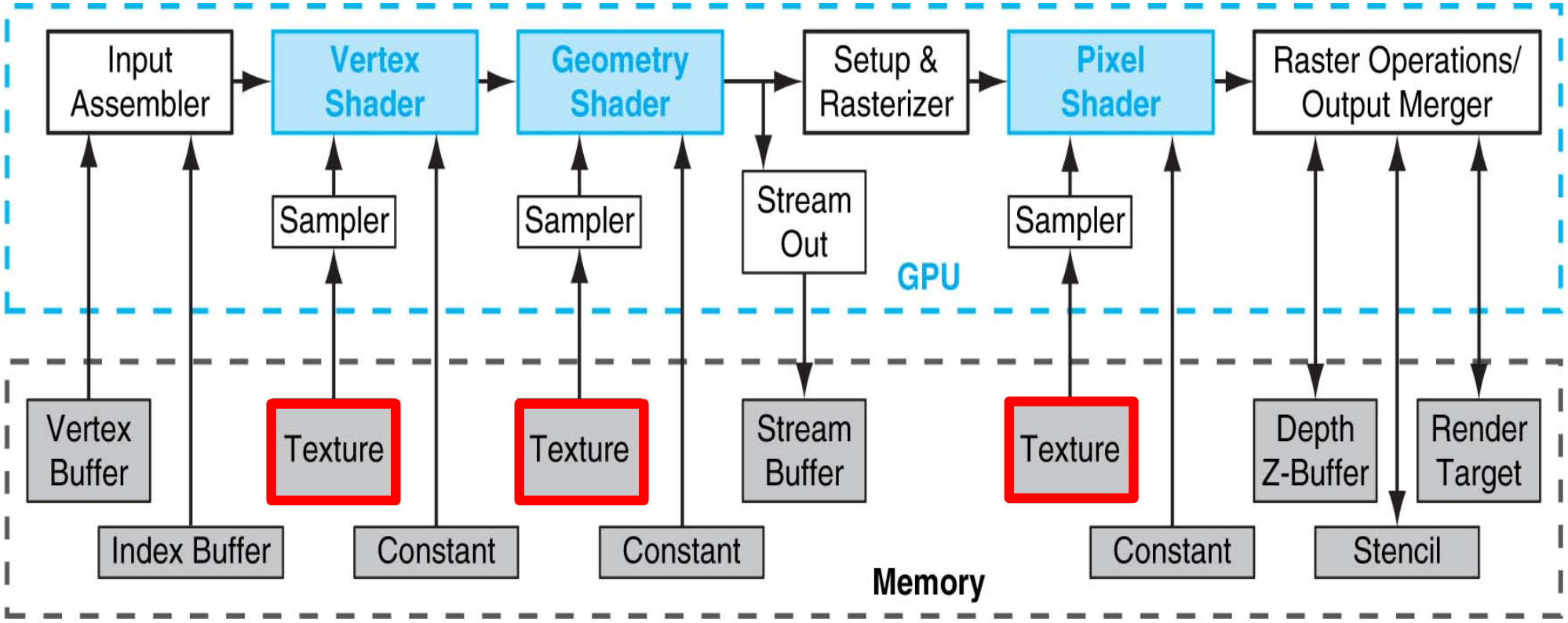
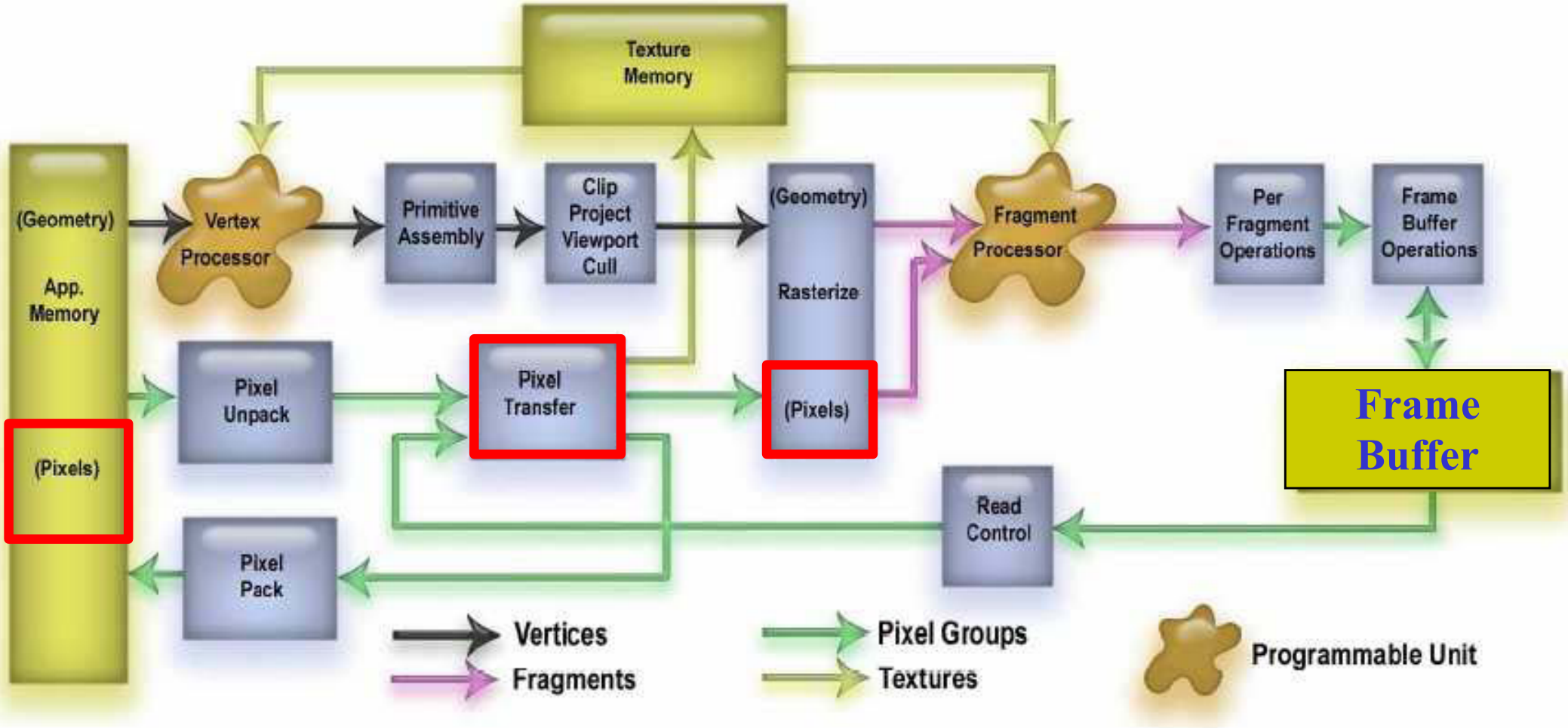


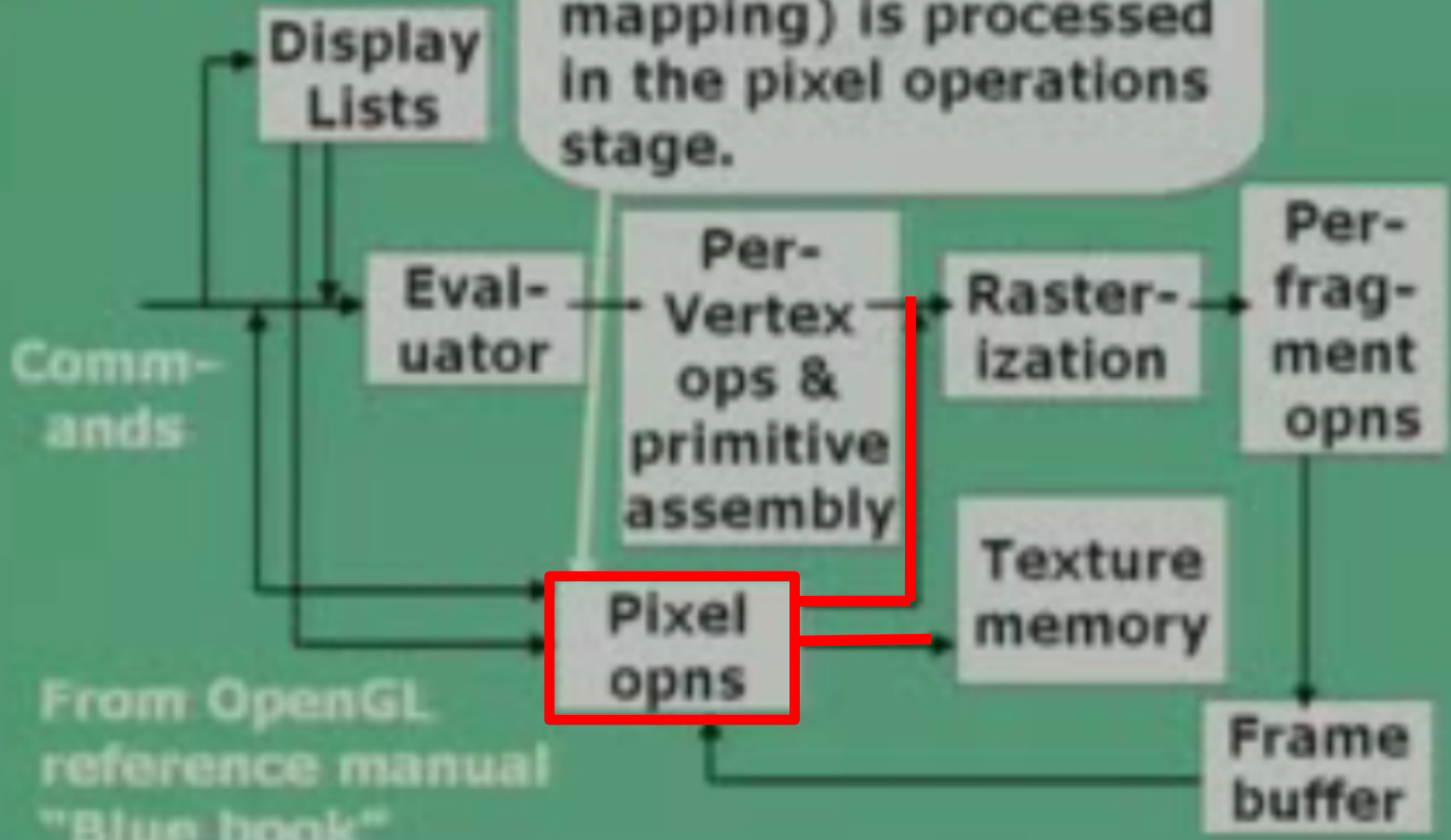
FIGURE A.3.1 Direct3D 10 graphics pipeline. Each logical pipeline stage maps to GPU hardware or to a **GPU** processor. **Programmable shader stages are blue, fixed-function blocks are white, and memory objects are grey. Each stage processes a vertex, geometric primitive, or pixel in a streaming dataflow fashion.**

OpenGL Graphics Pipeline



OpenGL Operation

Input data can be in the form of pixels (image for texture mapping) is processed in the pixel operations stage.



From OpenGL reference manual "Blue book"

Buffers

Chapter 8.1

Objectives

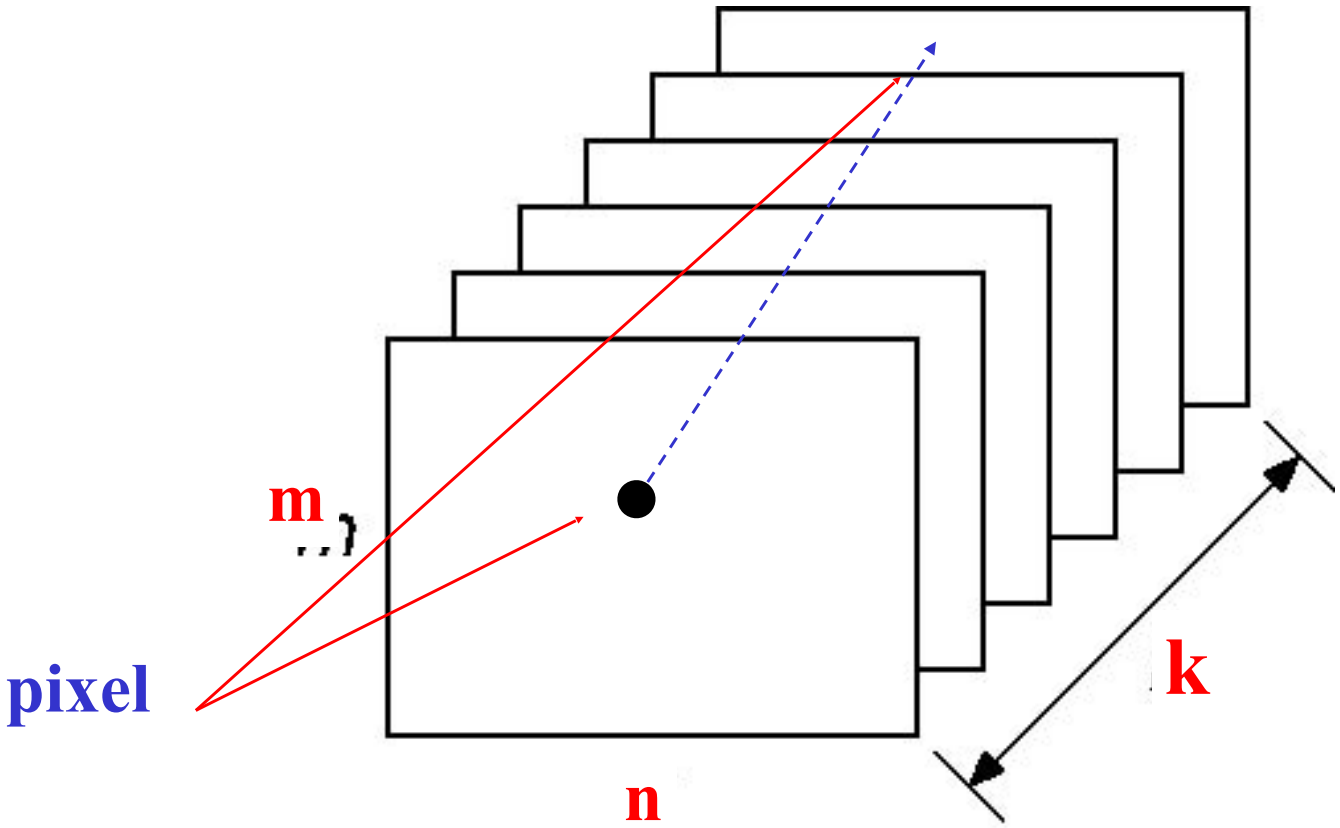
- Introduce additional **OpenGL** buffers
- Learn to read and write buffers
- Learn to use blending

Objectives

- Introduce additional **OpenGL** buffers
- Learn to read and write buffers
- Learn to use blending

Buffer

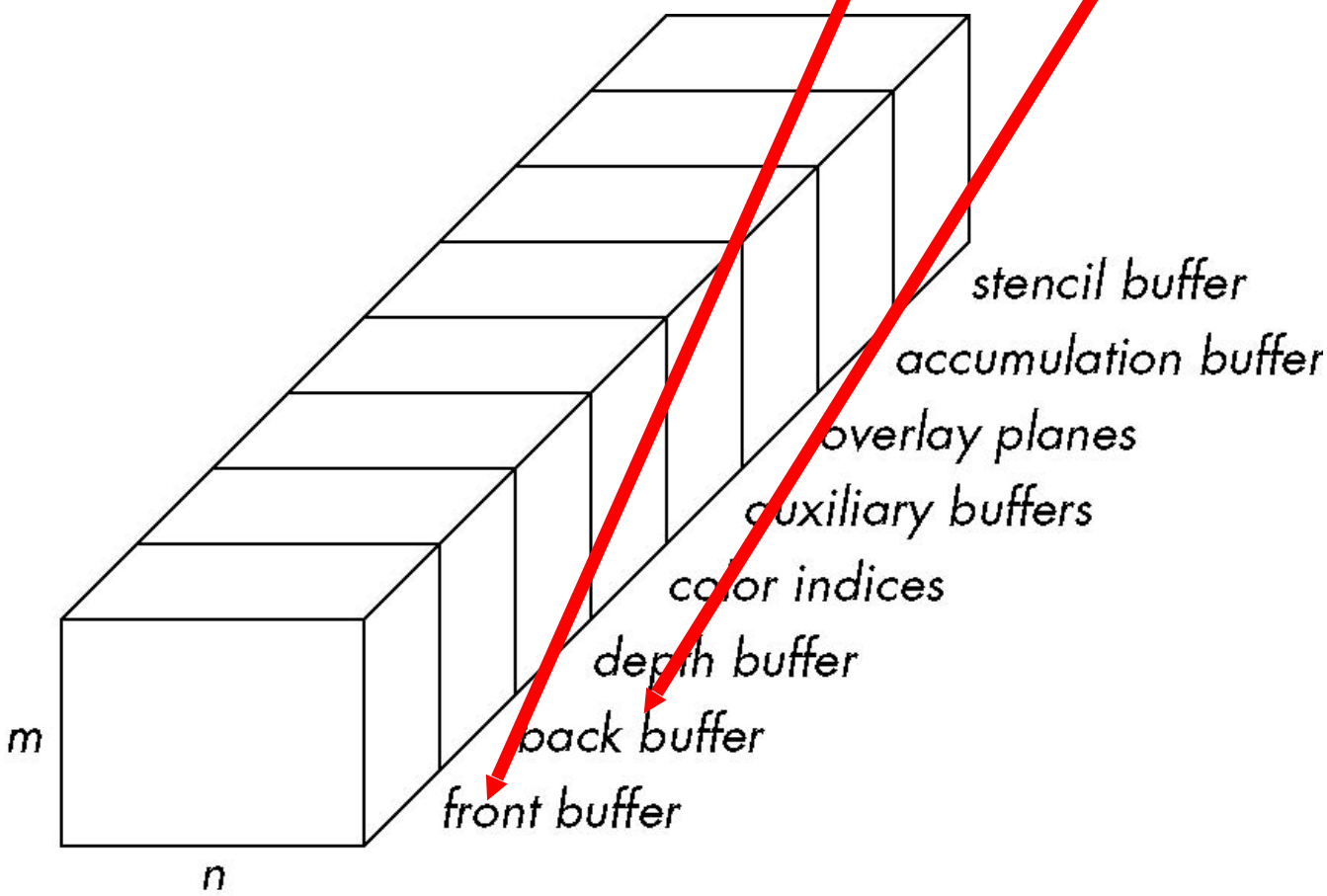
Define a **buffer** by its **spatial resolution** (**n** x **m**) and its **depth** (or **precision**) **k**, the number of **bits/pixel**



OpenGL Frame Buffer

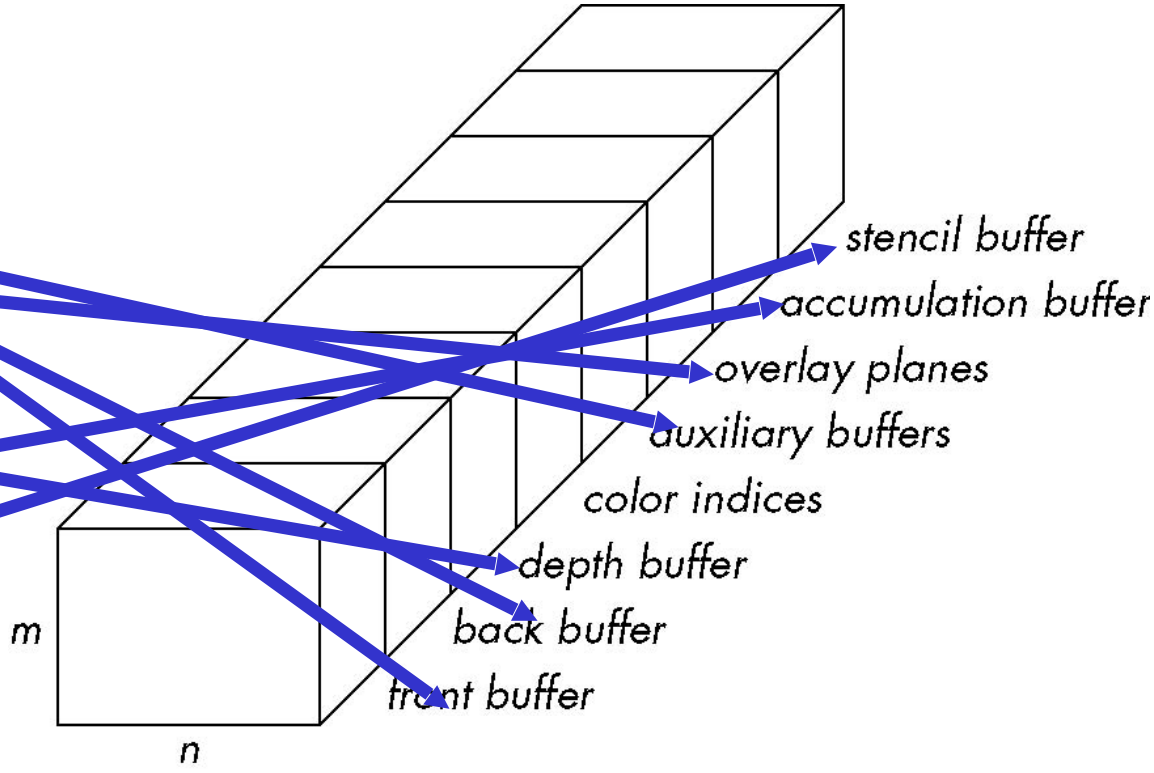
- If we consider the whole frame buffer, the resolution of the frame buffer, n and m, matches the spatial resolution of the display.

Thus, if a frame buffer has 32 bits each for its front and back color buffers, each RGBA color component is stored with a precision of 8 bits.



OpenGL Buffers

- Color **buffers** can be **displayed**
  - Front
  - Back
  - Auxiliary
  - Overlay
- Depth
- Accumulation
  - High resolution buffer
- Stencil
  - Holds masks



Objectives

- Introduce additional OpenGL buffers
- Learn to **read** and **write** **Buffers**

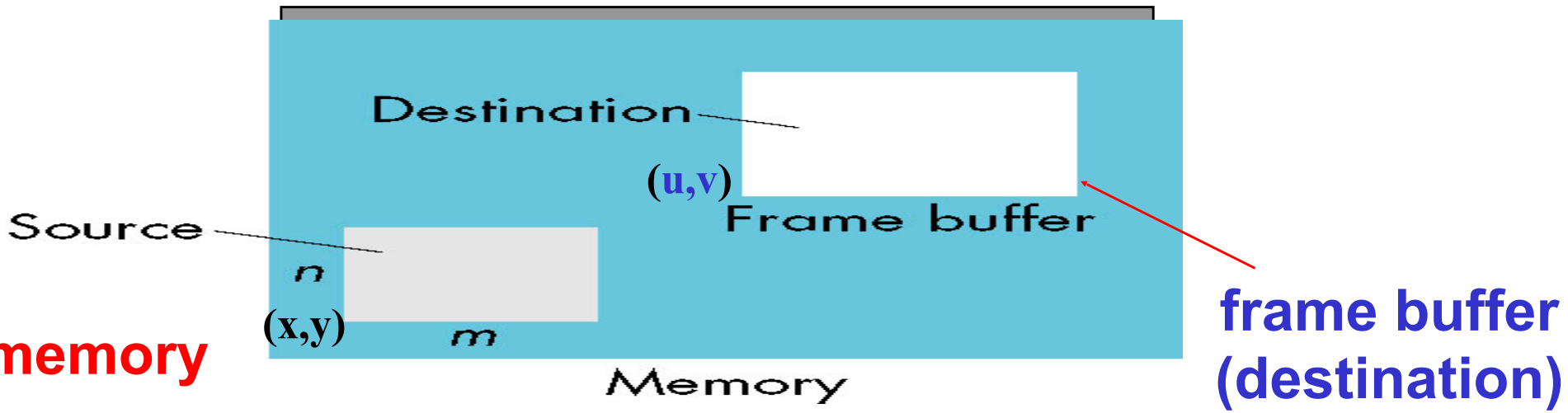
  Chapter 8.3
- Learn to use blending

Writing in Buffers (GPU)

- Conceptually, we can consider **all of memory** as a large **two-dimensional array of pixels**
- We read and write rectangular **block of pixels**

  Bit block *transfer* (***bitblt***) *operations*

- The **frame buffer** is part of this **memory**



writing into **frame buffer**

write_ block(**source**, n, m, **x**, **y**, **destination**, **u**, **v**);

Writing in Buffers

- Conceptually, we can consider **all of memory** as a large **two-dimensional array of pixels**
- We read and write rectangular block of pixels

***Bit block** transfer (**bitblt**) operations (**raster operations**)*

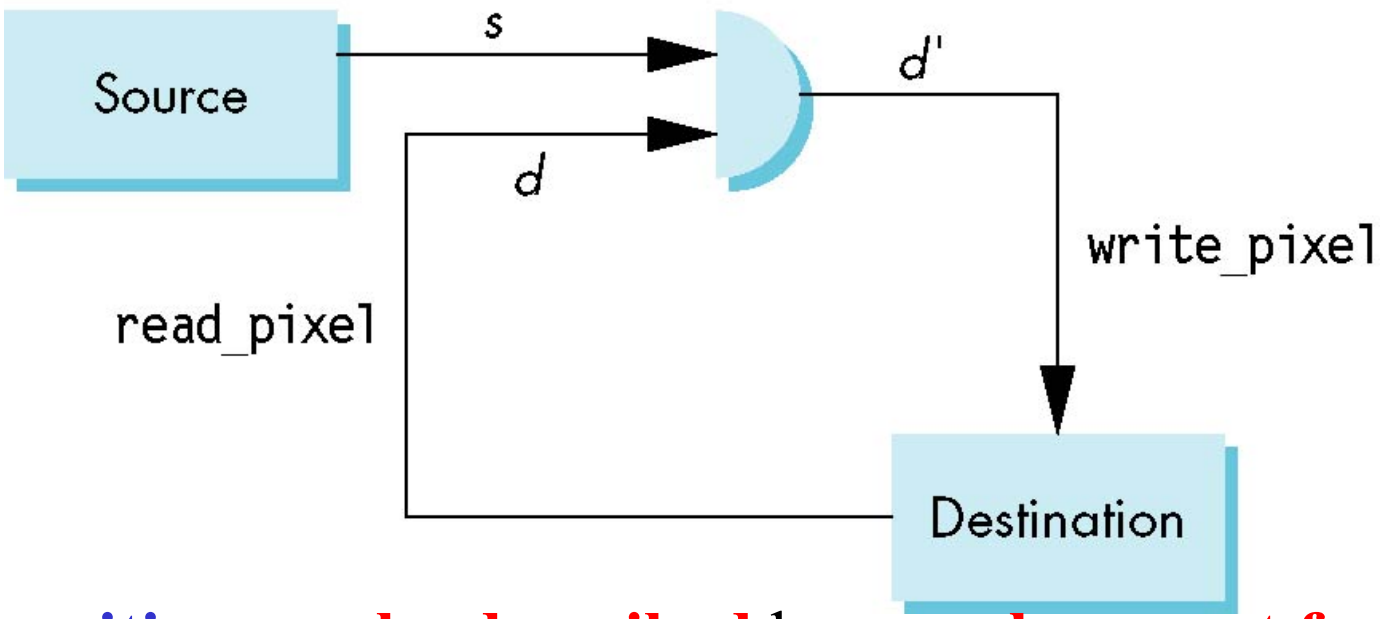
glClear operation changes the values of all pixels in a buffer

Note that, from the hardware perspective, the type of processing involved has none of the characteristics of the processing of **geometric objects**. Consequently, the **hardware that optimizes bitblt operations** has a completely different architecture from the **geometric pipeline**. Thus, the **OpenGL** architecture contains both a **geometry pipeline** and a **pixel pipeline**, each of which is usually implemented separately.

Writing Model

The usual concept of a **write** to memory is **replacement**. The execution of a statement in a C program such as **y = x;** results in the value at the location where **y is stored being replaced with the value at the location of x.**

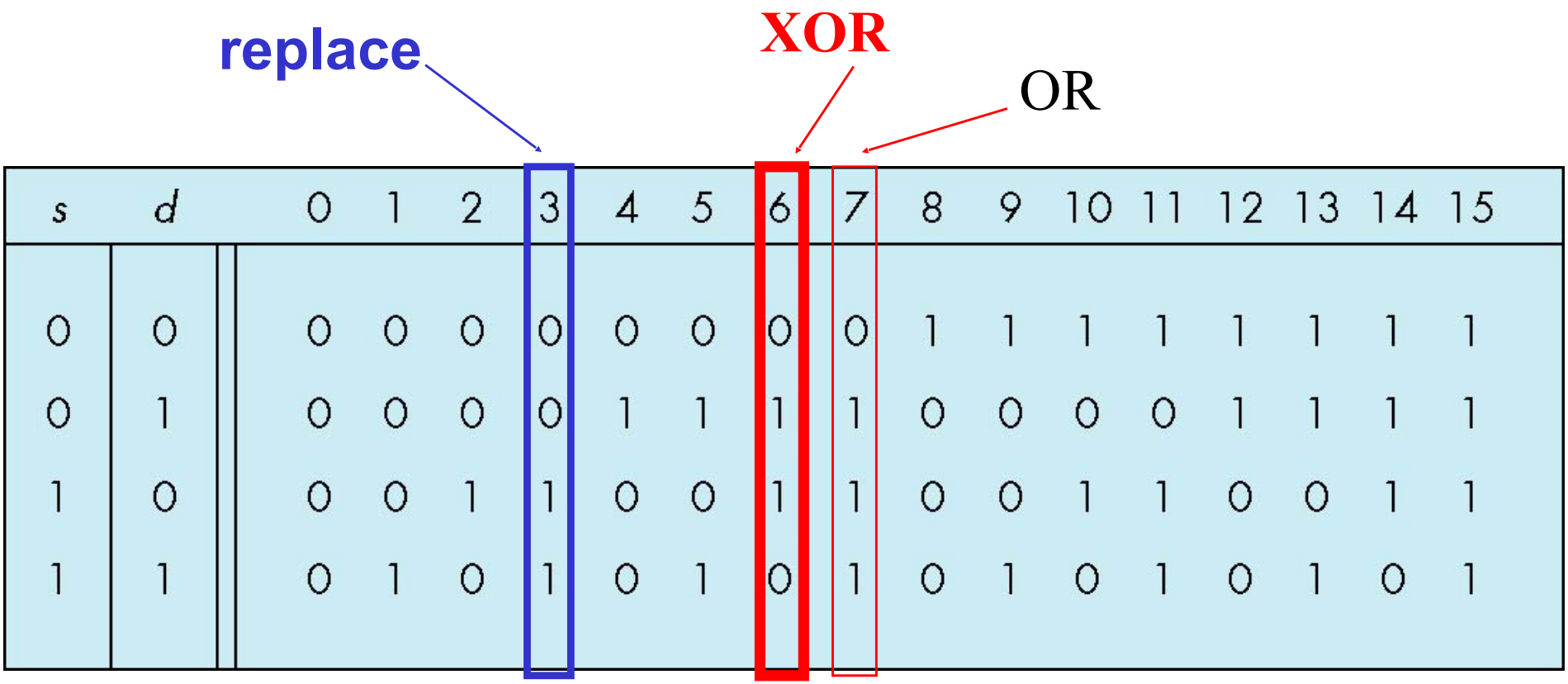
Read destination **pixel** before **writing** source



Then **writing can be described** by a **replacement function f such** that **d' ← f (d, s)**

Bit Writing Modes

- Source **s** and destination **d** bits are combined **bitwise**
- 16 possible **functions** (one per column in table)



s	*d*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

XOR mode

- Recall from Chapter 3 that we can use **XOR** by enabling logic operations and selecting the **XOR write mode**
- **XOR** is especially useful for **swapping blocks of memory** such as **menus** that are stored off screen

If S represents screen and M represents a menu the sequence

$$S \leftarrow S \oplus M$$
$$M \leftarrow S \oplus M$$
$$S \leftarrow S \oplus M$$

swaps the **S** and M

In response to a **mouse click**, a **menu appears**, covering a portion of the screen. After the **user indicates an action from the menu**, the **menu disappears**, and the **area of the screen** that it covered is returned to that **areas original state**.

CLASS PARTICIPATION 1!
(Next Slide)

Name: _____

Total score: [ ]

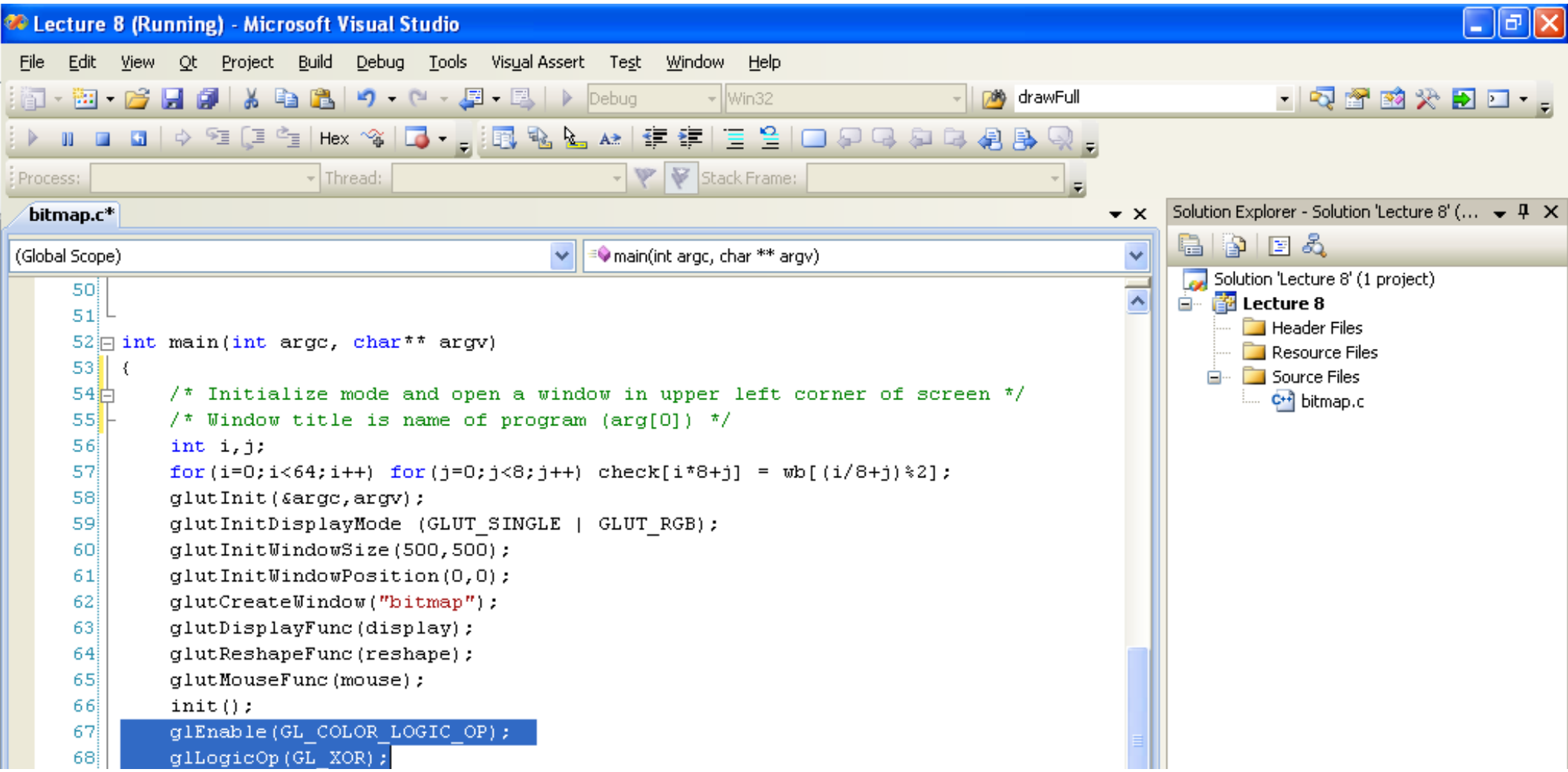
Class PARTICIPATION on Lecture 8.doc **ANSWER SHEET**

(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

1. (25 points) Download bitmap.c from CANVAS and build the Visual Studio 2019 C++ Project.

Self Graded - correctly



If **S** represents screen and M represents a menu the sequence

$$\mathbf{S} \leftarrow \mathbf{S} \oplus \mathrm{M}$$

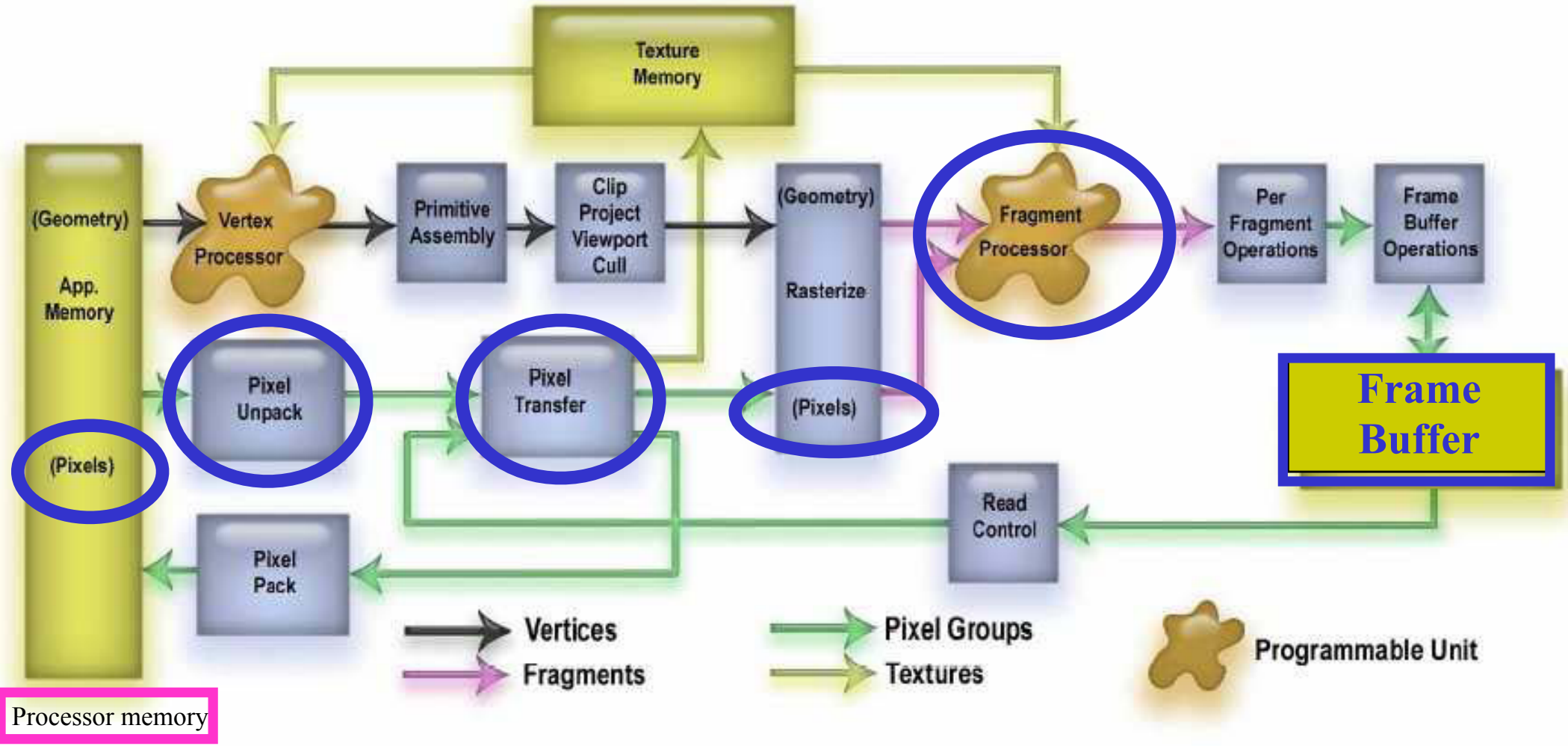
$$\mathrm{M} \leftarrow \mathbf{S} \oplus \mathrm{M}$$

$$\mathbf{S} \leftarrow \mathbf{S} \oplus \mathrm{M}$$

swaps the **S** and M

glBitmap(**64**, **64**, 0.0, 0.0, 0.0, 0.0, **check**);

OpenGL Graphics Pipeline



The Pixel Pipeline

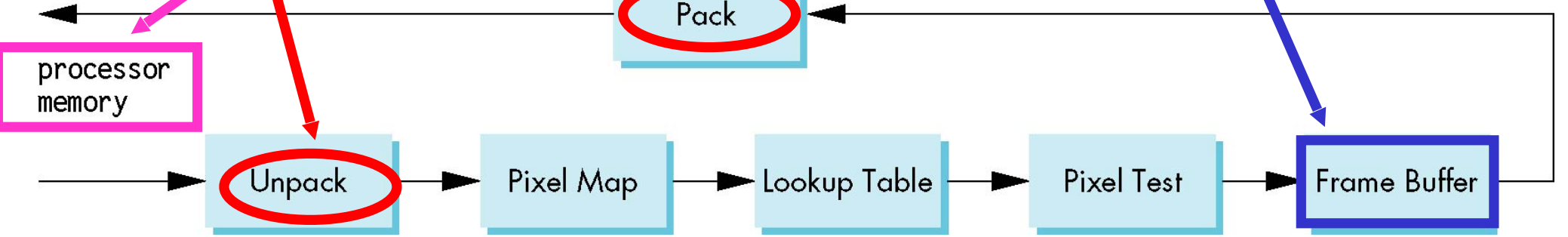
OpenGL has a separate pipeline for pixels

Writing pixels involves

- Moving pixels from processor memory to the frame buffer
- Format conversions
- Mapping, Lookups, Tests

Reading pixels

- Format conversion



Raster Position (internal cursor)

OpenGL maintains a ***raster position (int)*** as part of the state

Set by **glRasterPos*()**

```
glRasterPos3f(x, y, z);
```

```
36  void mouse(int btn, int state, int x, int y)
37  {
38
39      if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
40          {
41          glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
42          glRasterPos2i(x, ww-y);
43          glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
44          glFlush();
45          }
46  }
47
```

The **raster position (x,y,z,w)** is a **geometric entity**

- Passes through **geometric pipeline**
- Eventually yields a **2D position in screen coordinates**
- This position in the **frame buffer** is **where** the **next raster primitive is drawn**

Solution 'Lecture ...

(1 of 1 project)

endencies

es

Class View | Property ...

bit.c

(Global Scope) | mouse(int btn, int state, int x, int y)

```
31      glMatrixMode(GL_PROJECTION);
32      glLoadIdentity();
33      gluOrtho2D (0.0,   (GLfloat) w, 0.0, (GLfloat) h);
34      ww=w;
35      glMatrixMode(GL_MODELVIEW);
36      glLoadIdentity();
37  }
38
39  void mouse(int btn, int state, int x, int y)
40  {
41
42      if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
43          {
            glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
            glRasterPos2i(x, ww-y);
            glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
47          glFlush();
48          }
49  }
50
51
52  int main(int argc, char** argv)
53  {
54
55  /* Initialize mode and open a window in upper left corner of screen */
56  /* Window title is name of program (arg[0]) */
57
58      int i,j;
59      for(i=0;i<64;i++) for(j=0;j<8;j++) check[i*8+j] = wb[(i/8+j)%2];
60      glutInit(&argc,argv);
```

glRasterPos3f(x, y, z);

Buffer Selection

OpenGL can **Draw into** or **Read from** any of the color buffers (**front**, **back**, **auxiliary**, **overlay**)

Default to the **back buffer**

Change with **`glDrawBuffer`** and **`glReadBuffer`**

Note that **format of the pixels in the frame buffer** is different from that of **processor memory** and these **two types of memory reside in different places**

- **Need packing and unpacking**
- **Drawing and Reading can be slow**

Bitmaps

glBitmap(width, height, xo, yo, xi, yi, bitmap);

OpenGL treats **1-bit pixels** (***bitmaps***) differently from **multi-bit pixels** (***pixelmaps***)

Bitmaps are **masks** that determine if the corresponding pixel in the **frame buffer** is drawn with the *present* ***raster color*** at the **current raster position**

0 ⇒ color unchanged

1 ⇒ color changed based on **writing mode**

Bitmaps are useful for **raster text**

GLUT font: `GLUT_BIT_MAP_8_BY_13`

Solution 'Lecture ...

(1 of 1 project)

endencies

es

Class View | Property ...

bit.c

(Global Scope) | mouse(int btn, int state, int x, int y)

```
31      glMatrixMode(GL_PROJECTION);
32      glLoadIdentity();
33      gluOrtho2D (0.0,   (GLfloat) w, 0.0, (GLfloat) h);
34      ww=w;
35      glMatrixMode(GL_MODELVIEW);
36      glLoadIdentity();
37  }
38
39  void mouse(int btn, int state, int x, int y)
40  {
41
42      if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
43          {
44          glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
45          glRasterPos2i(x, ww-y);
46          glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
47
48
49  }
50
51
52  int main(int argc, char** argv)
53  {
54
55  /* Initialize mode and open a window in upper left corner of screen */
56  /* Window title is name of program (arg[0]) */
57
58      int i,j;
59      for(i=0;i<64;i++) for(j=0;j<8;j++) check[i*8+j] = wb[(i/8+j)%2];
60      glutInit(&argc,argv);
```

glBitmap(width,height, xo, yo, xi, yi, bitmap);

Raster Color

- Same as drawing color set by `glColor*()`
- Fixed by last call to `glRasterPos*()`

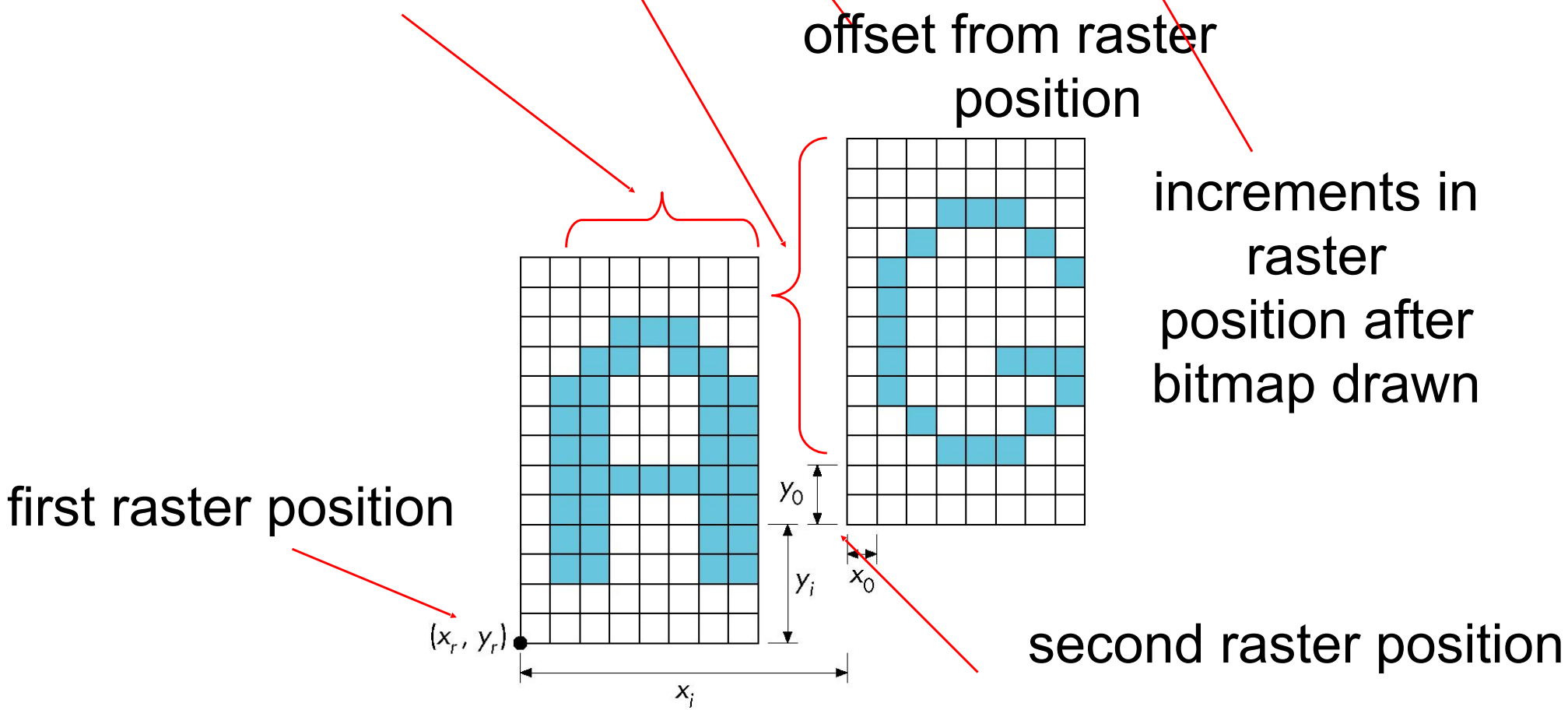
```
glColor3f(1.0, 0.0, 0.0);
 glRasterPos3f(x, y, z);
glColor3f(0.0, 0.0, 1.0);
      glBitmap(…….
   glBegin(GL_LINES);
     glVertex3f(…..)
```

- **Geometry drawn in blue**
- Ones in **bitmap** use a drawing color of red

See Homework 7!

Drawing Bitmaps

```
glBitmap(width, height, x0, y0, xi, yi, bitmap)
```



The width and height are in bits. The floating- point numbers x_o and y_o give an offset relative to the **current raster position**, and give the position where the lower-left corner of the bitmap is to start. x_i and y_i are floating-point numbers used to **move the raster position** after the bitmap is drawn.

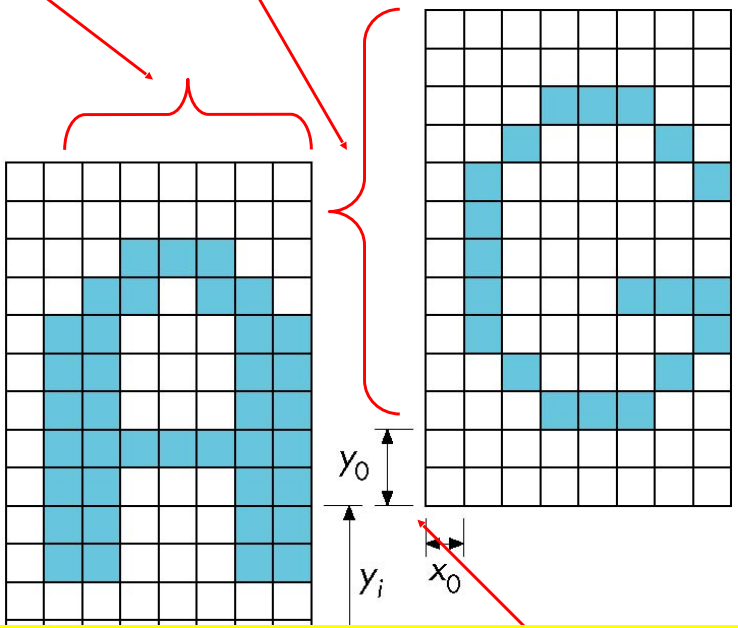
Drawing Fonts

```
glBitmap(width, height, x0, y0, xi, yi, bitmap)
```

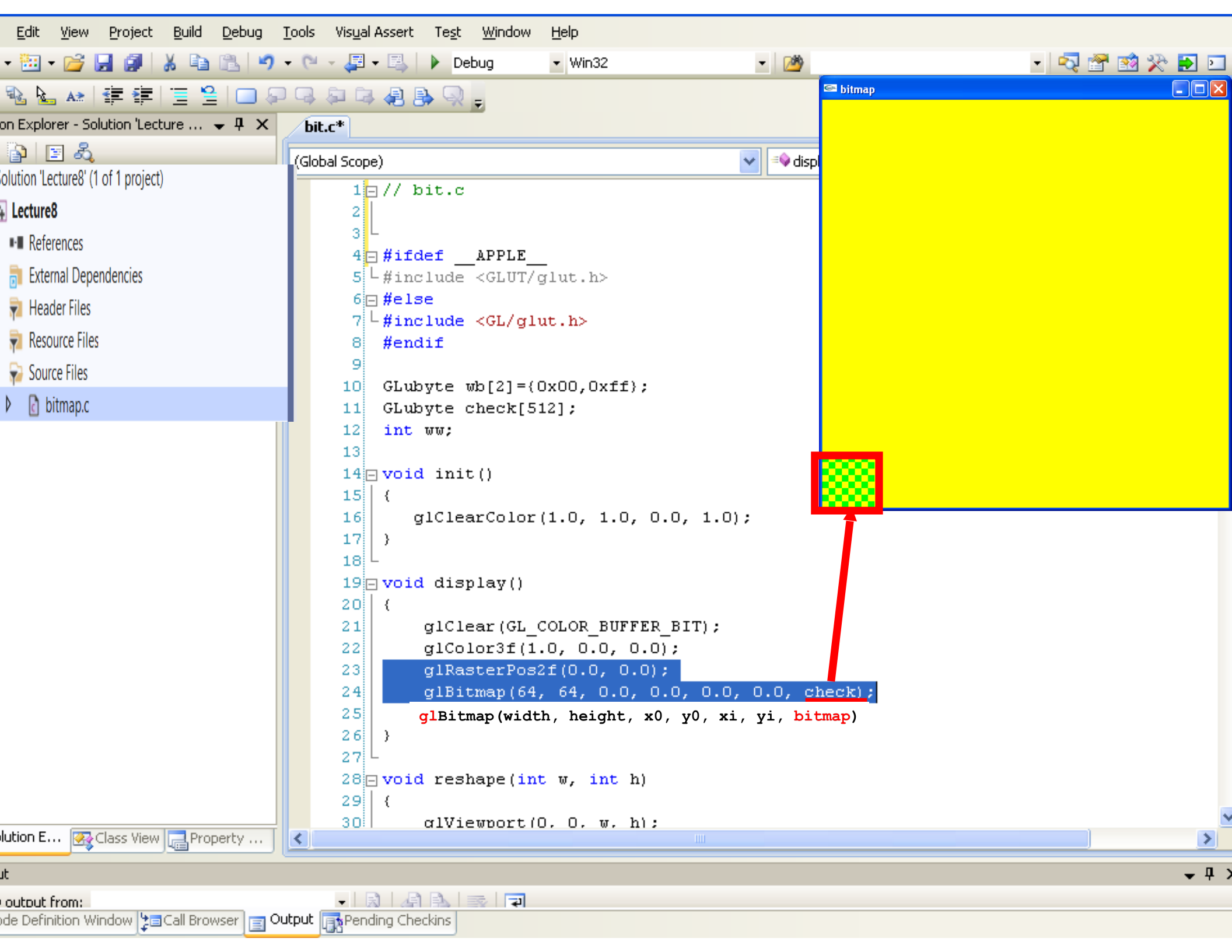
offset from raster position

increments in raster position after bitmap drawn

first raster position



Please ADD text to the S'COOL BUS!!!!

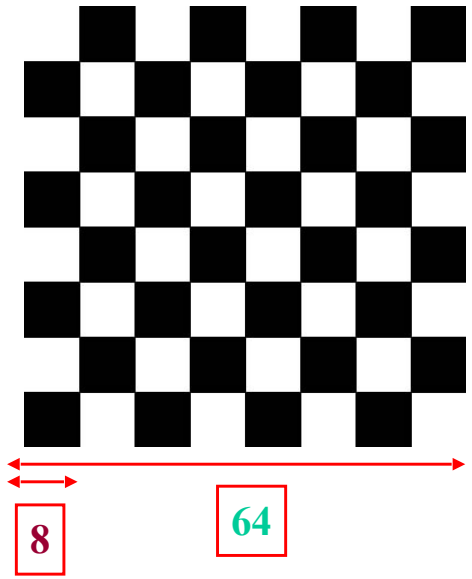

Edit View Project Build Debug Tools Visual Assert Test Window Help
Debug
Win32
bitmap
bit.c*
(Global Scope)
// bit.c
#ifdef __APPLE__
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
GLubyte wb[2]={0x00,0xff};
GLubyte check[512];
int ww;
void init()
{
glClearColor(1.0, 1.0, 0.0, 1.0);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 0.0, 0.0);
glRasterPos2f(0.0, 0.0);
glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
glBitmap(width, height, x0, y0, xi, yi, bitmap)
}
void reshape(int w, int h)
{
glViewport(0, 0, w, h);
Lecture8
References
External Dependencies
Header Files
Resource Files
Source Files
bitmap.c
Class View
Property ...
Call Browser
Output
Pending Checkins

Bitmap Example: Checker Board

```
GLubyte wb[2] = {0 x 00, 0 x ff};
GLubyte check[512];

int i, j;
for (i=0; i<64; i++)
   for (j=0; j<64, j++)
      check[i*8+j] = wb[(i/8+j)%2];

glBitmap( 64, 64, 0.0, 0.0, 0.0, 0.0, check);
```



Pixel Maps

OpenGL works with rectangular **arrays of pixels** called **Pixel Maps** or **Images**

Pixels are in byte (8 bit) chunks

- Luminance (gray scale) images 1 floating point/**pixel**
- RGB 3 bytes/**pixel**

Three functions

- **Draw pixels**: **processor memory** to **frame buffer**
- **Read pixels**: **frame buffer** to **processor memory**
- **Copy pixels**: **frame buffer** to **frame buffer**

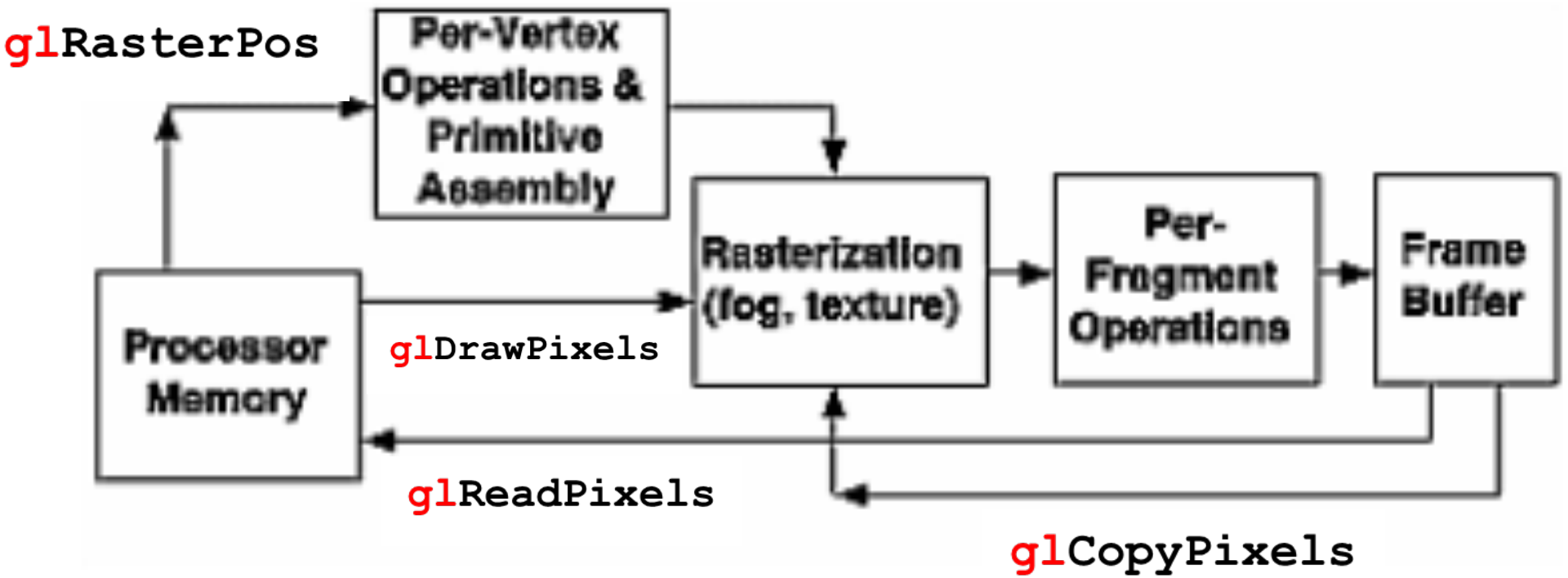


Figure 8-3 : Simplistic Diagram of Pixel Data Flow

OpenGL Pixel Functions

```
glReadPixels(x,y,width,height,format,type,myimage)
```

start pixel in frame buffer

size

type of pixels

type of image

pointer to processor memory

```
GLubyte myimage[512][512][3];
glReadPixels(0,0, 512, 512, GL_RGB, GL_UNSIGNED_BYTE, myimage);
```

Read pixels: frame buffer to processor memory

OpenGL Pixel Functions

```
glDrawPixels(width,height,format,type,myimage)
```

starts at **raster position**

Draw pixels: **processor memory** to **frame buffer**

Image Formats

- We often work with **images in a standard format** (JPEG, TIFF, GIF)

- How do we **read/write** such images with **OpenGL**?

- **No support in OpenGL**
  - **OpenGL knows nothing of image formats**
  - **Some code available on Web**
  - **Can write readers/writers for some simple formats in OpenGL**

CLASS PARTICIPATION 2!
(Next Slide)

Displaying a PPM (Portable Pixel Map) Image

- **PPM** is a very simple format
- Each **Image** file consists of a header followed by all the pixel data
- Header

P3
comment 1
comment 2
.
#comment n
rows columns maxvalue
pixels

Lecture 8 - Microsoft Visual Studio

File Edit View Qt Project Build Debug Tools Visual Assert Test Window Help

Debug | Win32 | drawFull

ppmdisplay.c

(Global Scope) | main(int argc, char ** argv)

```
82      for(i=0;i<nm;i++)
83      {
84          fscanf(fd,"%d %d %d",&red, &green, &blue );
85          image[3*nm-3*i-3]=red;
86          image[3*nm-3*i-2]=green;
87          image[3*nm-3*i-1]=blue;
88      }
89      printf("read image\n");
90      glutInit(&argc, argv);
91      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
92      glutInitWindowSize(n, m);
93      glutInitWindowPosition(0,0);
94      glutCreateWindow("ppm display image");
95      glutReshapeFunc(myreshape);
96      glutDisplayFunc(display);
97      glPixelTransferf(GL_RED_SCALE, s);
98      glPixelTransferf(GL_GREEN_SCALE, s);
99      glPixelTransferf(GL_BLUE_SCALE, s);
100     glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
101     glClearColor(1.0, 1.0, 1.0, 1.0);
102     glutMainLoop();
103 }
104
105
106
```

Solution Explorer - Solution 'Lecture 8' (1 proj...

Solution 'Lecture 8' (1 project)
- Lecture 8
  - Header Files
  - Resource Files
  - Source Files
    - ppmdisplay.c

Solution Explorer | Class View

Server Explorer

Reading the Header

```
FILE *fd;
int  k, nm;
char c;
int i;
char b[100];
float s;
int red, green, blue;
printf("enter file name\n");
scanf("%s", b);
fd = fopen(b, "r");
fscanf(fd,"%[^\n] ",b);
if(b[0]!='P'|| b[1] != '3'){
printf("%s is not a PPM file!\n", b);
exit(0);
}
printf("%s is a PPM file\n",b);
```

check for "**P3**" in first line

Reading the Header

```
fscanf(fd, "%c",&c);
   while(c == '#')
           {
 fscanf(fd, "%[^\n] ", b);
     printf("%s\n",b);
    fscanf(fd, "%c",&c);
           }
      ungetc(c,fd);
```

skip over comments by
looking for **#** in first column

Reading the Data

rows columns maxvalue

```
fscanf(fd, "%d %d %d", &n, &m, &k);
printf("%d rows  %d columns  max value= %d\n",n,m,k);

nm = n*m;
image=malloc(3*sizeof(GLuint)*nm);
s=255./k;
```

scale factor

```
for(i=0;i<nm;i++)
{
fscanf(fd,"%d %d %d",&red, &green, &blue );
image[3*nm-3*i-3]=red;
image[3*nm-3*i-2]=green;
image[3*nm-3*i-1]=blue;
}
```

Scaling the Image Data

We can scale **s** the image in the **pipeline**

```
glPixelTransferf(GL_RED_SCALE,   s);
glPixelTransferf(GL_GREEN_SCALE, s);
glPixelTransferf(GL_BLUE_SCALE,  s);
```

We may have to swap bytes (little endian, big endian) when we go from **processor memory** to the **frame buffer** depending on the processor. If so, we can use

```
glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
```

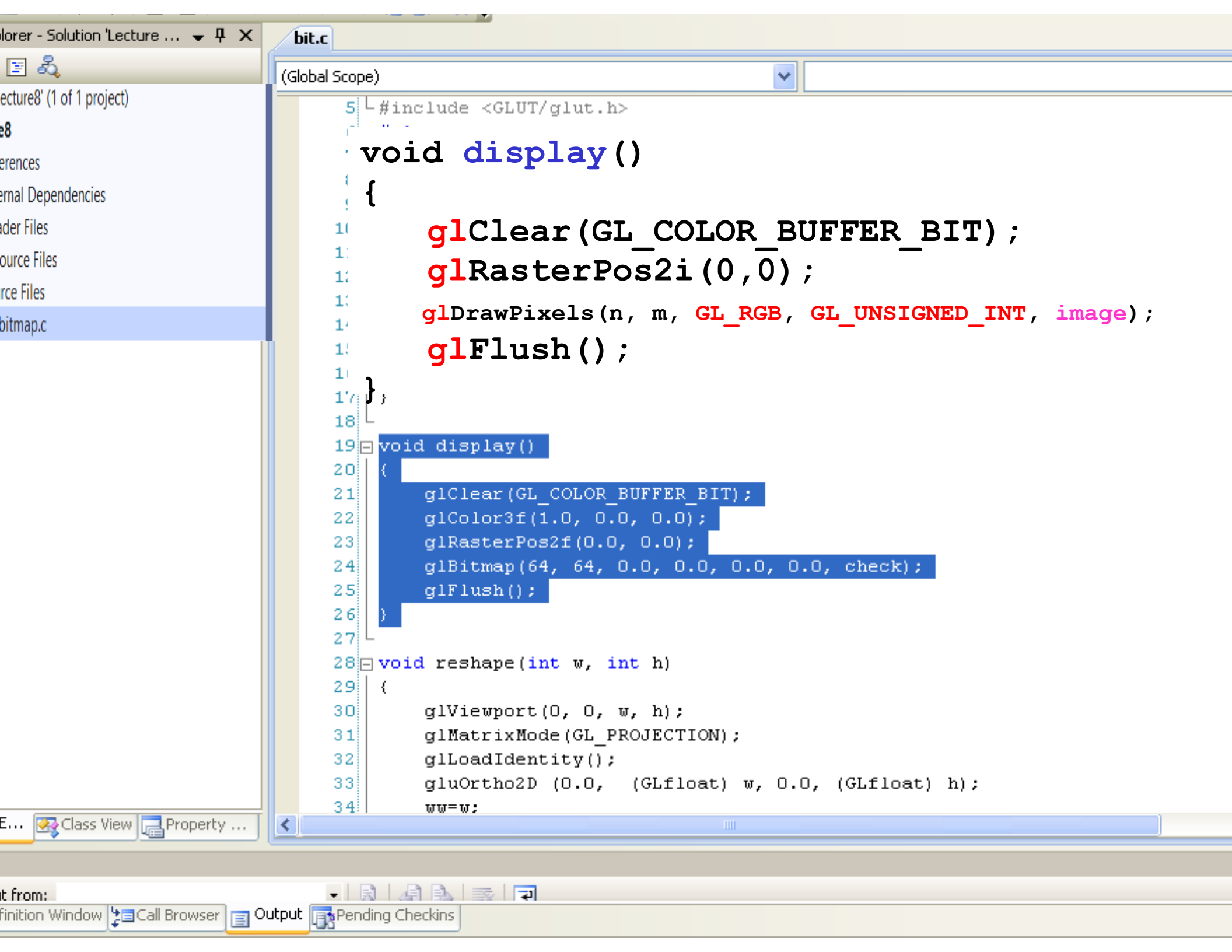
The display callback

```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glRasterPos2i(0,0);
    glDrawPixels(n, m, GL_RGB, GL_UNSIGNED_INT, image);
    glFlush();
}
```

```
glDrawPixels(width,height,format,type,myimage)
```

starts at **raster position**

Draw pixels: **processor memory** to **frame buffer**


lorer - Solution 'Lecture ...
ecture8' (1 of 1 project)
e8
erences
ernal Dependencies
ader Files
ource Files
rce Files
bitmap.c
bit.c
(Global Scope)
#include <GLUT/glut.h>
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glRasterPos2i(0,0);
glDrawPixels(n, m, GL_RGB, GL_UNSIGNED_INT, image);
glFlush();
};
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 0.0, 0.0);
glRasterPos2f(0.0, 0.0);
glBitmap(64, 64, 0.0, 0.0, 0.0, 0.0, check);
glFlush();
}
void reshape(int w, int h)
{
glViewport(0, 0, w, h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D (0.0, (GLfloat) w, 0.0, (GLfloat) h);
ww=w;
Class View
Property ...
t from:
finition Window
Call Browser
Output
Pending Checkins

```
f:\COSC 4370 ======= SUMMER 2011\LECTURES\CLASS PARTICIPATION ON LECTURE 8\Le
enter file name
robot2.ppm
P3 is a PPM file
 CREATOR: XV Version 3.10a  Rev: 12/29/94
 CREATOR: XV Version 3.10a  Rev: 12/29/94
 CREATOR: XV Version 3.10a  Rev: 12/29/94  Quality = 75, Smoothing = 0
256 rows  256 columns  max value= 255
read image
finished display
```

Solution 'Lecture 8' (1 project)

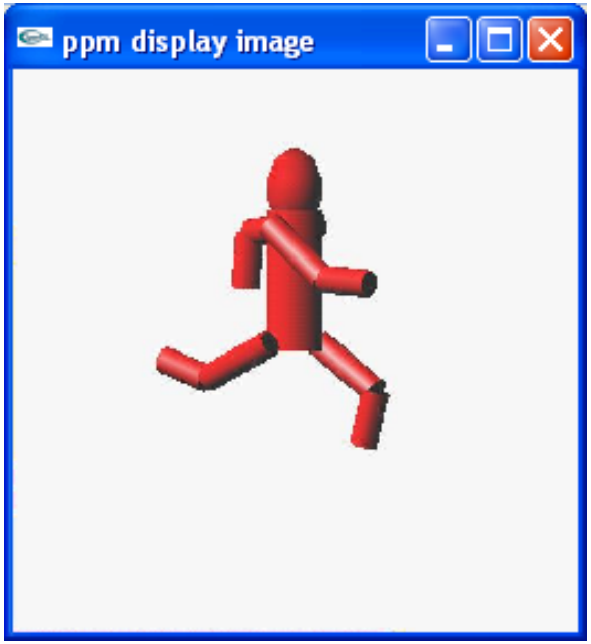
- Lecture 8
  - Header Files
  - Resource Files
  - Source Files
    - ppmdisplay.c

```
F:\COSC 4370 ======
enter file name
```

robot2.ppm
PPM File
781 KB

```
F:\COSC 4370 ==
enter file name
robot2.ppm
```

```
78      s=255./k;
79
80
81
82      for(i=0;i<nm;i++)
83      {
84           fscanf(fd,"%d %d %d",&red, &green, &blue );
85           image[3*nm-3*i-3]=red;
86           image[3*nm-3*i-2]=green;
87           image[3*nm-3*i-1]=blue;
88      }
        printf("read image\n");
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
        glutInitWindowSize(n, m);
        glutInitWindowPosition(0,0);
        glutCreateWindow("ppm display image");
        glutReshapeFunc(myreshape);
96      glutDisplayFunc(display);
97      glPixelTransferf(GL_RED_SCALE, s);
98      glPixelTransferf(GL_GREEN_SCALE, s);
99      glPixelTransferf(GL_BLUE_SCALE, s);
100     glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
101
102     glPixelStorei(GL_UNPACK_SWAP_BYTES, GL_TRUE);
103 }
104
105
```



Solution Explorer | Class View

Pending Checkins

Check In | Comments

Call Stack | Breakpoints | Output | Pending Checkins

Mapping Methods

Chapters 8.6- 8.7

Objectives

- Introduce Mapping Methods
  - **Texture Mapping**
  - **Environment Mapping**
  - **Bump Mapping**
- Consider basic strategies
  - **Forward vs backward mapping**
  - **Point sampling vs area averaging**

Objectives

- Introduce Mapping Methods
  - **Texture Mapping**
  - **Environment Mapping**
  - **Bump Mapping**
- Consider basic strategies
  - Forward vs backward mapping
  - Point sampling vs area averaging

The Limits of Geometric Modeling

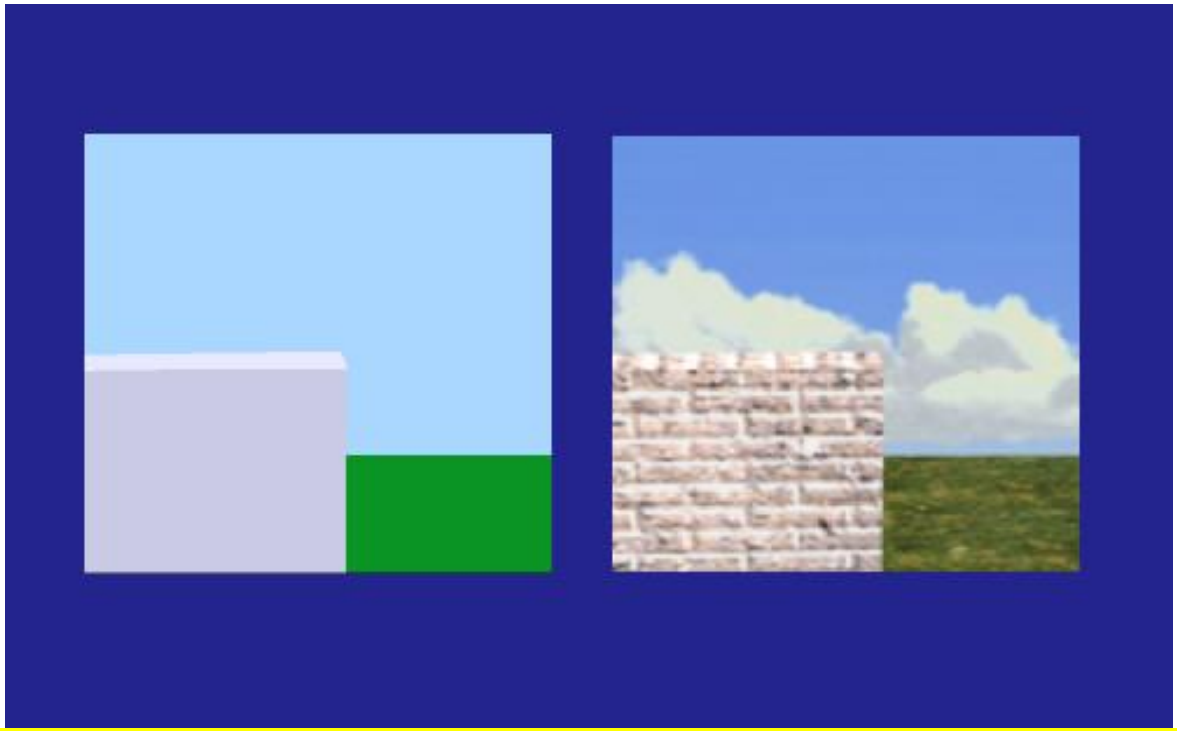
Although **graphics cards** can **render over 10 million polygons per second**, that number is insufficient for **many phenomena**

Clouds

Grass

Terrain

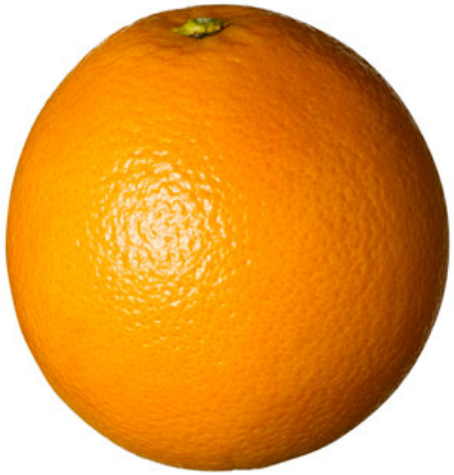
Skin



Geometric COMPLEXITY low Illusion of COMPLEX geometry!

When creating image detail, it is cheaper to employ mapping techniques that it is to **use myriads of tiny polygons.** The image on the right portrays a brick wall, a lawn and the sky. In actuality the wall was modeled as a rectangular solid, and the lawn and the sky were created from rectangles. **The entire image contains eight polygons.** Imagine the number of polygon it would require to model the blades of grass in the lawn! **Texture mapping** creates the appearance of grass without the cost of rendering thousands of

Modeling an Orange



Consider the problem of modeling an orange

Start with an orange-colored sphere

Too simple

Replace sphere with a more complex shape

- **Does not capture surface characteristics (small dimples)**
- **Takes too many polygons to model all the dimples**

Modeling an Orange

- **Take a picture of a real orange, scan it, and "paste" onto simple geometric model**

  This process is known as **texture mapping**

- **Still might not be sufficient because resulting surface will be smooth**

  Need to change local shape

  Bump mapping

Three Types of Mapping

- **Texture Mapping**

  Uses **Images** to fill **inside of polygons**

- **Environment (reflection mapping)**

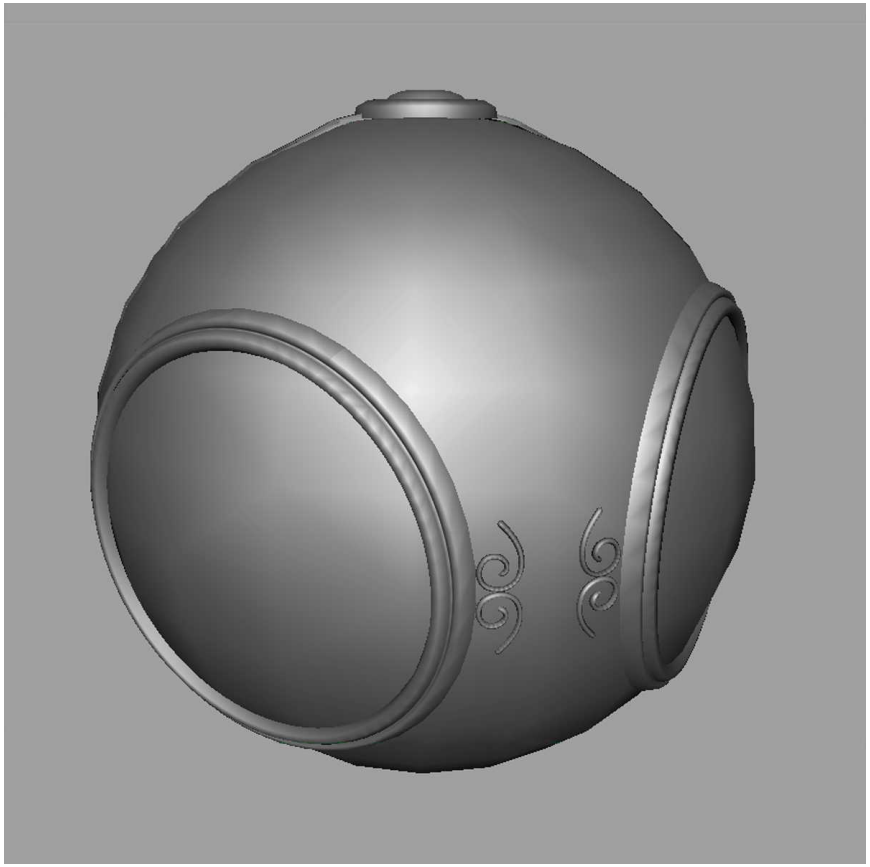
  Uses a **Picture** of the environment for **texture maps**

  Allows simulation of **highly specular surfaces**

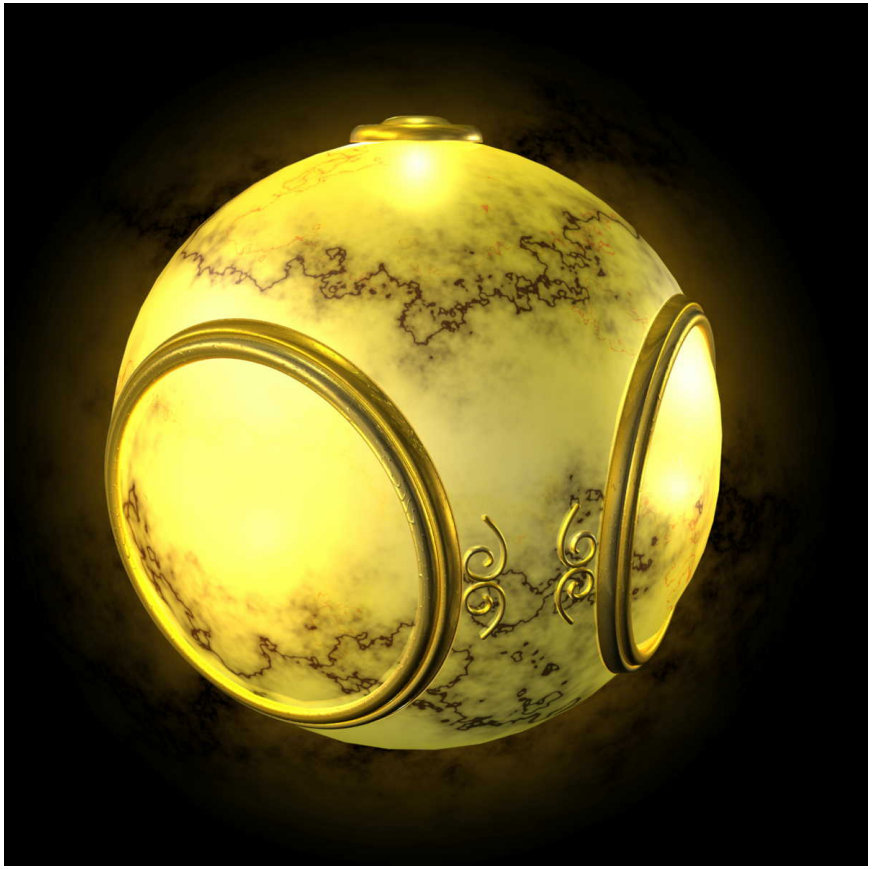
- **Bump mapping**

  Emulates **altering normal vectors** during the **rendering** process

Texture Mapping

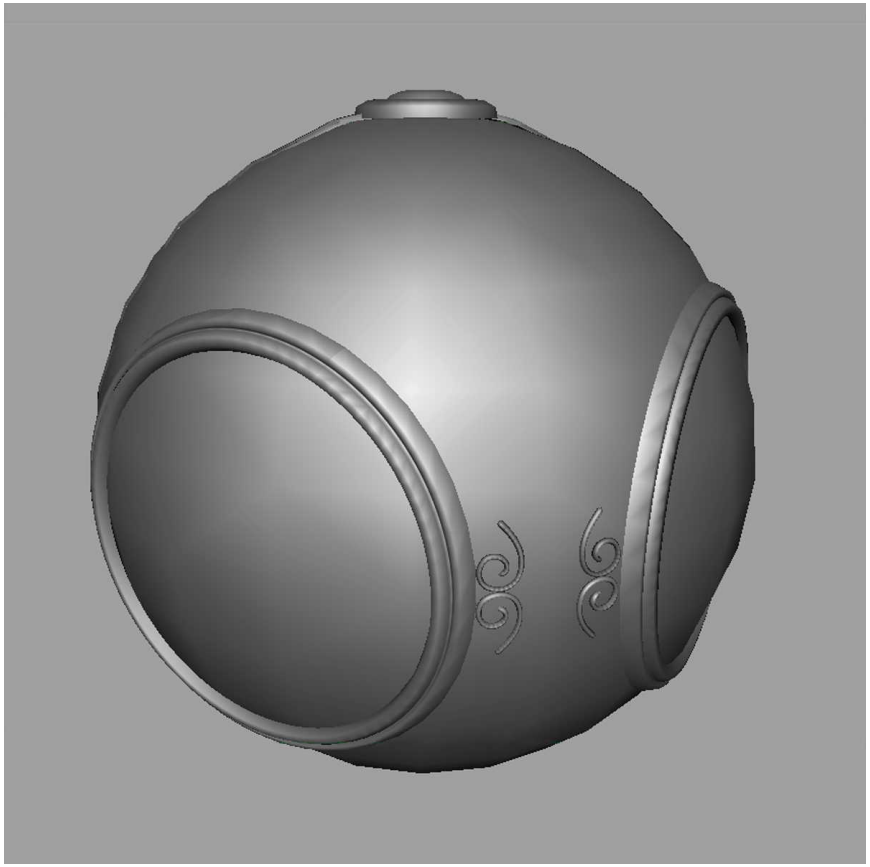


geometric model



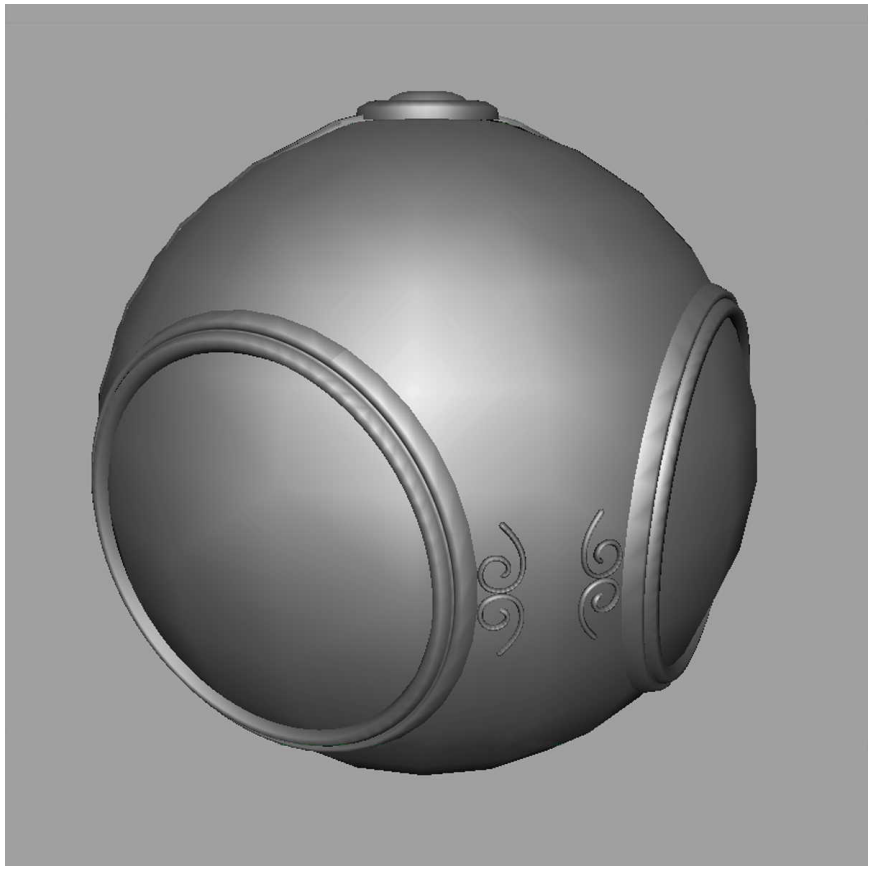
texture mapped

Environment Mapping

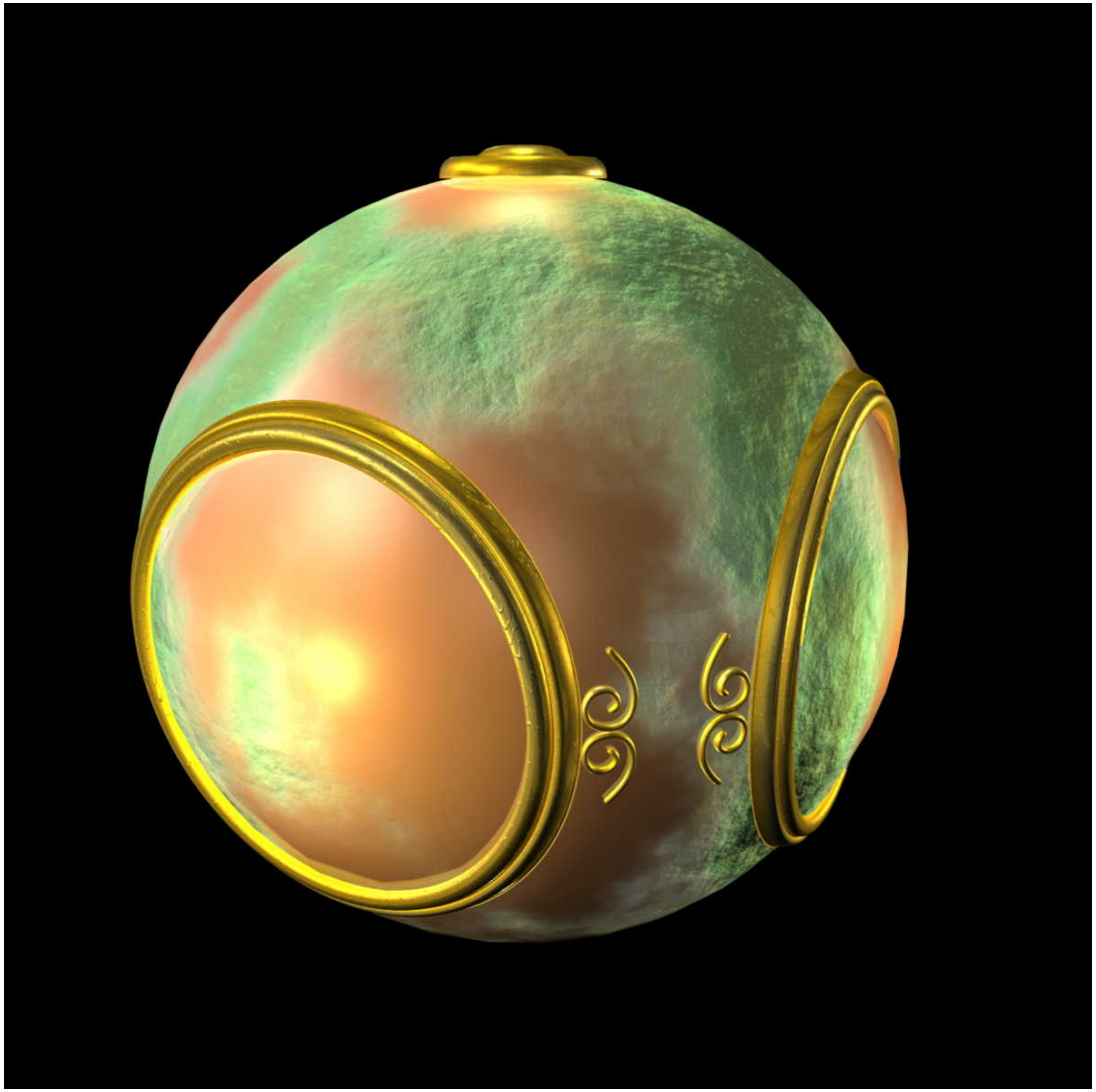


geometric model

Bump Mapping



geometric model



Bump Mapping

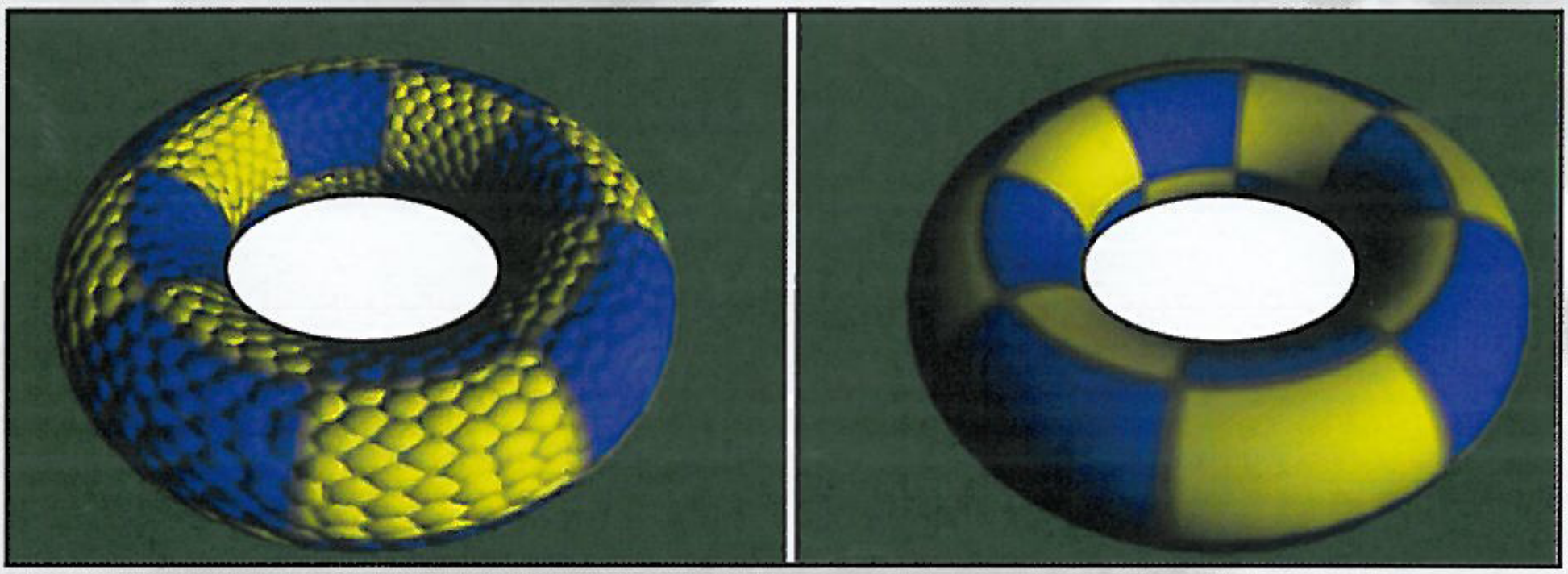


PLATE 22: An example of bump mapping.

Texture Mapping

Reflection Mapping

Displacement Mapping

Reflection Mapping

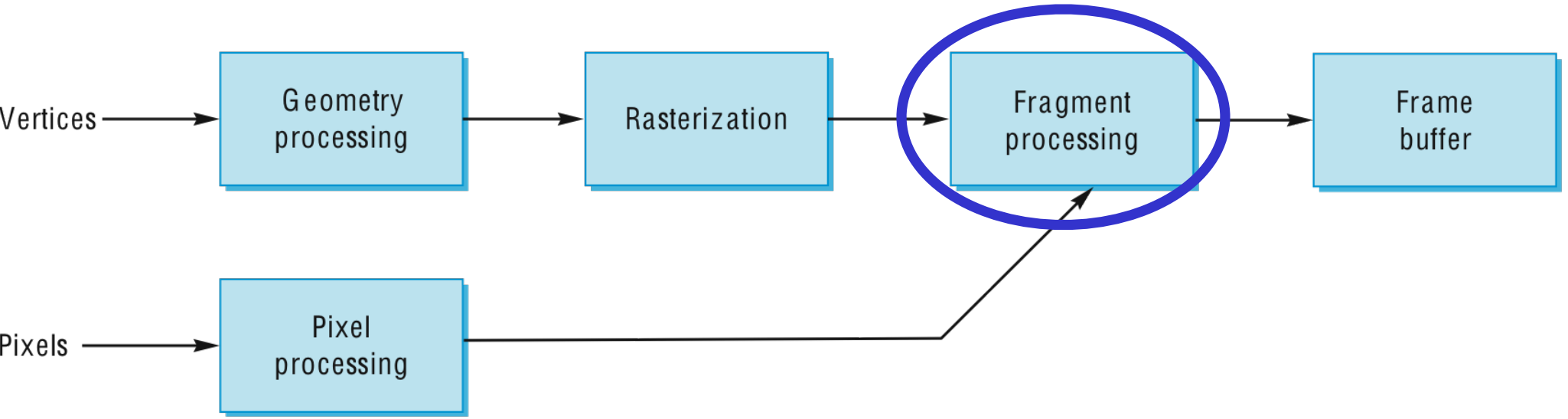
Objectives

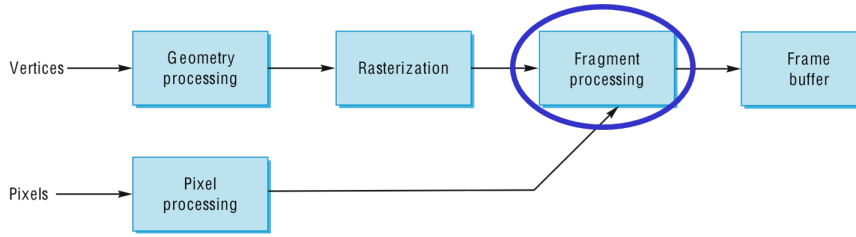
- Introduce Mapping Methods
  - Texture Mapping
  - Environment Mapping
  - Bump Mapping

- Consider basic strategies
  - **Forward** vs **backward mapping**
  - **Point sampling** vs **area averaging**

Where does mapping take place?

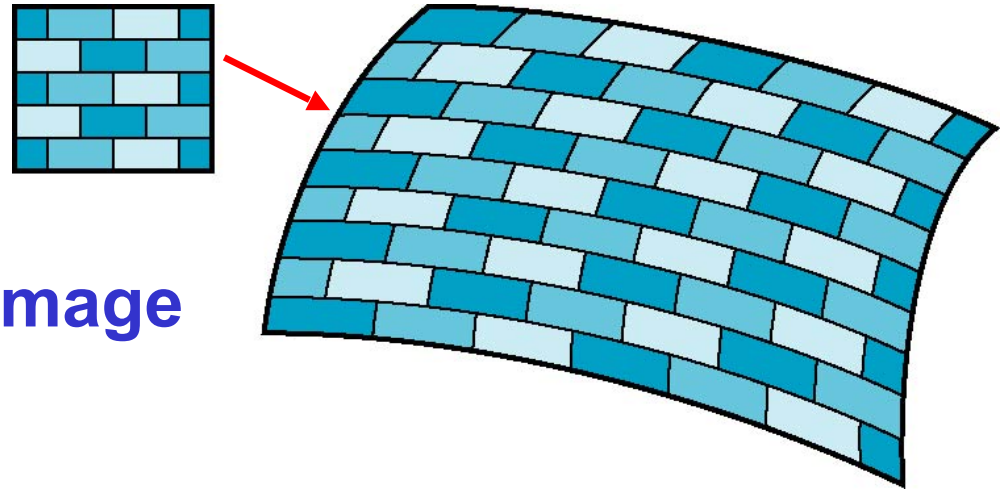
- **Mapping techniques** are implemented at the **end of the rendering pipeline**
- Very efficient because **few polygons** make it past the **clipper**





Is it simple?

- Although the **idea is simple** -- **map an Image (2D) to a surface (3D)** - there are 3 or 4 coordinate systems involved



2D image

3D surface

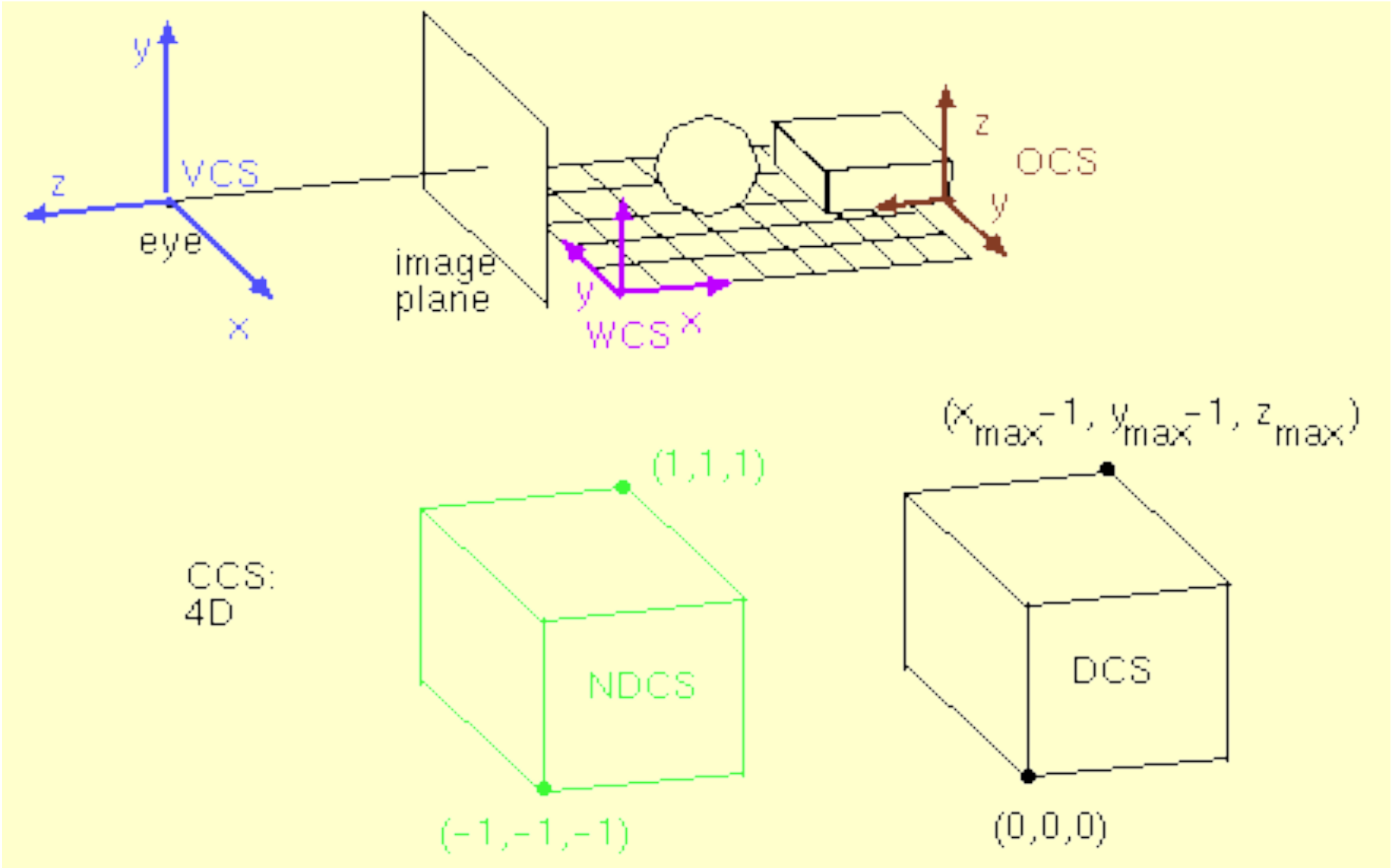
Alter the shading of individual fragments as part of the fragment processing

1, 2, 3 dimensional Textures

- Image (1D) to a curve (1D) - color the curve
- Image (2D) to a surface (2D) - texture the surface
- Image (3D) to a solid (3D) - sculpt an object

We will concentrate on 2 dimensional textures

Stages of Vertex Transformations



Coordinate Systems

- **Parametric coordinates**

  May be used to **model curves and surfaces**

- **Texture coordinates**

  Used to identify **points in the Image to be mapped**

- **Object or World Coordinates**

  Conceptually, where the **mapping** takes place

- **Window Coordinates**

  Where the **final image is really produced**

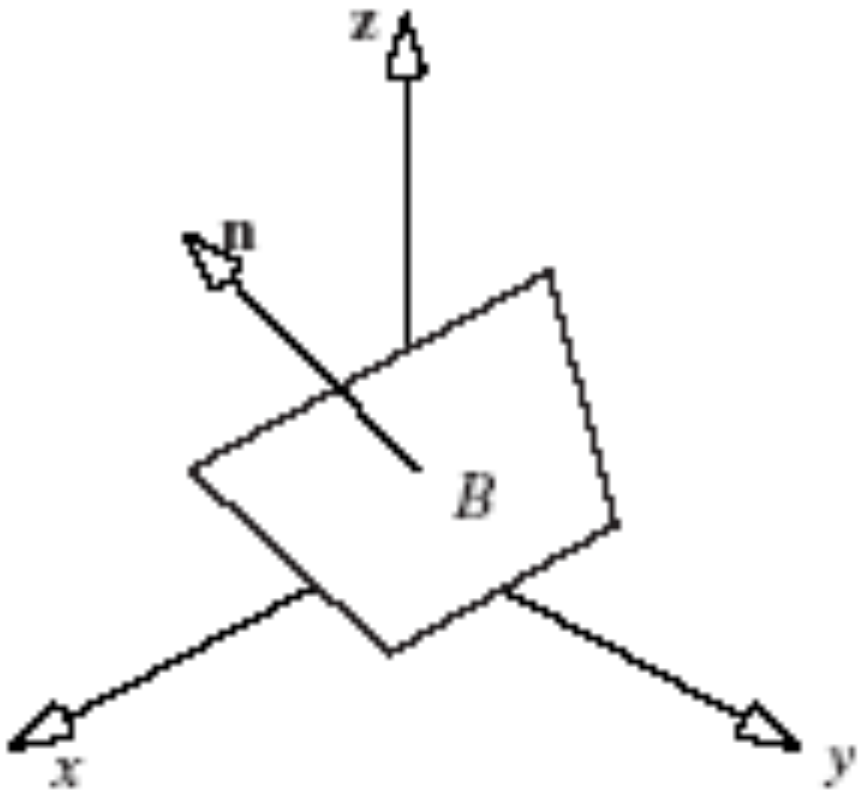
CLASS PARTICIPATION 3!
(Next Slide)

Representing Planes: Point-Normal Form

Point-normal form: $\mathbf{n} \cdot (P - B) = 0$

where B is a given point on the plane,

and $P = (x, y, z, 1)^T$



Planes: Parametric Form

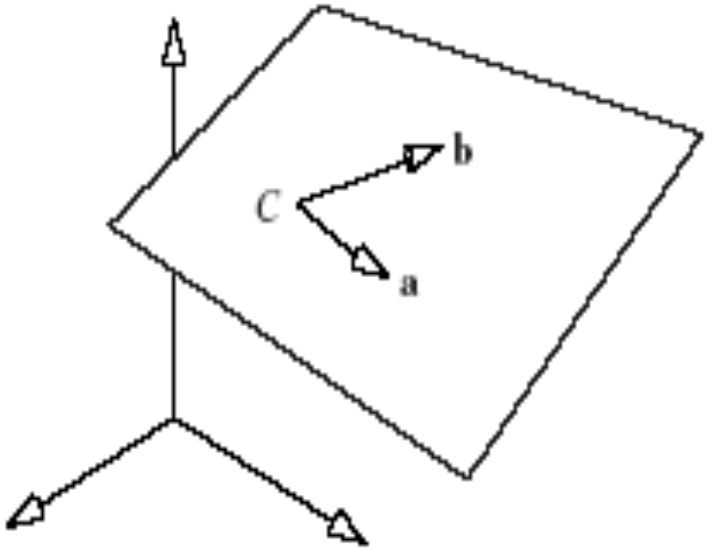
A plane can be infinite in 2 directions, semi-infinite, or finite.

Parametric form: requires 3 non-collinear points on the plane A, B, and C.

P(s, t) = C + s**a** + t**b**, where

a = A – C and

b = B – C.



$-\infty \le$ **s** $\le \infty$ and $-\infty \le$ **t** $\le \infty$: infinite plane.

$0 \le$ **s** ≤ 1 and $0 \le$ **t** ≤ 1: a finite plane, or **patch**.

Planes: Parametric Form

- We can rewrite

$$P(s,t) = C + s\mathbf{a} + t\mathbf{b}$$

where $\mathbf{a} = A - C$

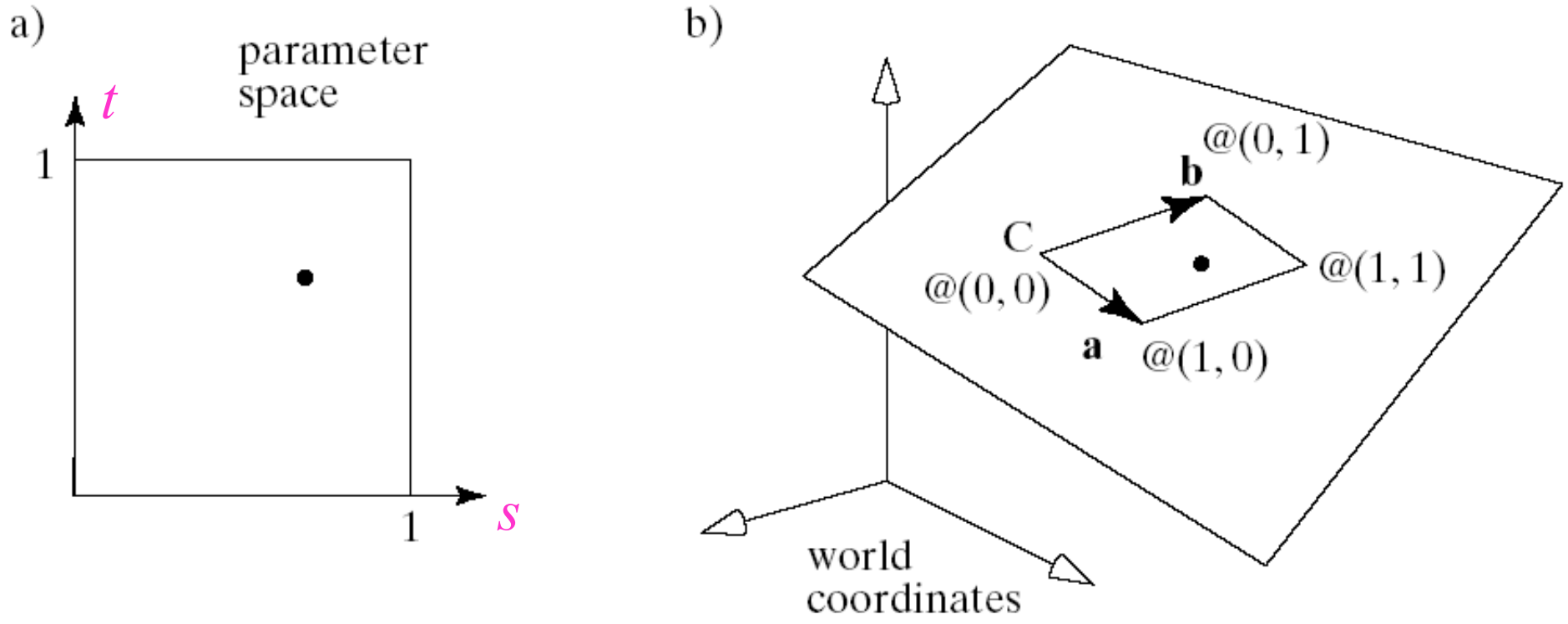
and $\mathbf{b} = B - C$

as an affine combination of points:

$$P(s, t) = s\,A + t\,B + (1 - s - t)C$$

Planes: Parametric Form

The figure shows the available range of *s* and *t* as a ***square in parameter space***, and the ***patch*** that results from this restriction in **object space**.

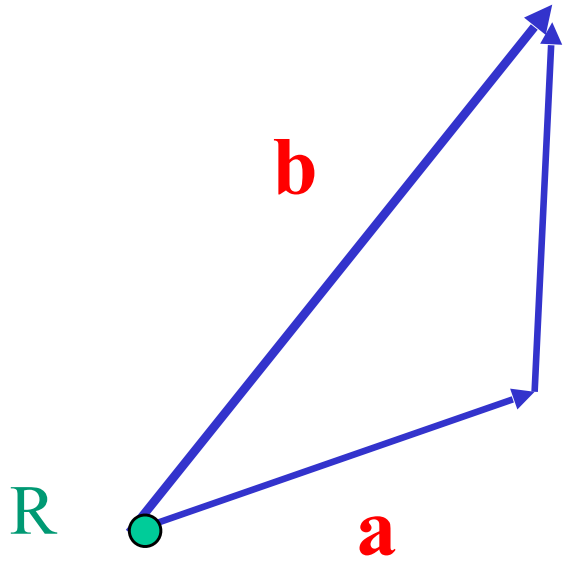


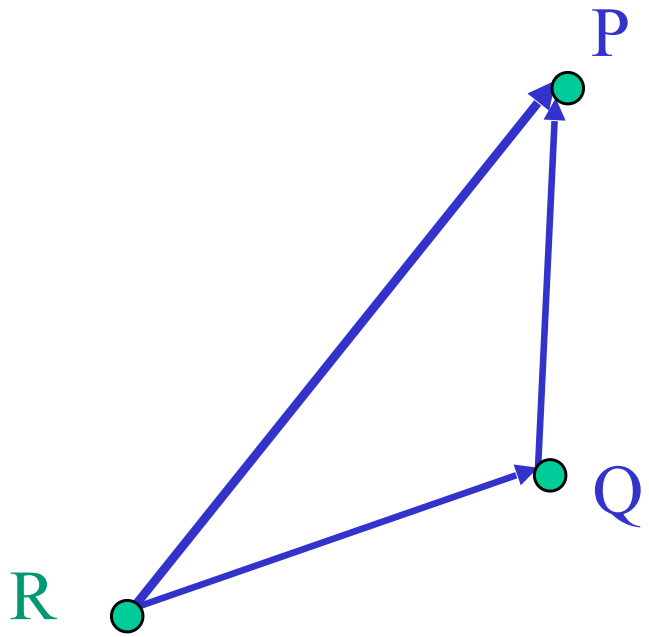
Planes – Parametric Form

A **plane** with two parameters u, v can be defined by

1 point R **and two vectors** (**a** and **b**) **or** by

3 points P, Q, R

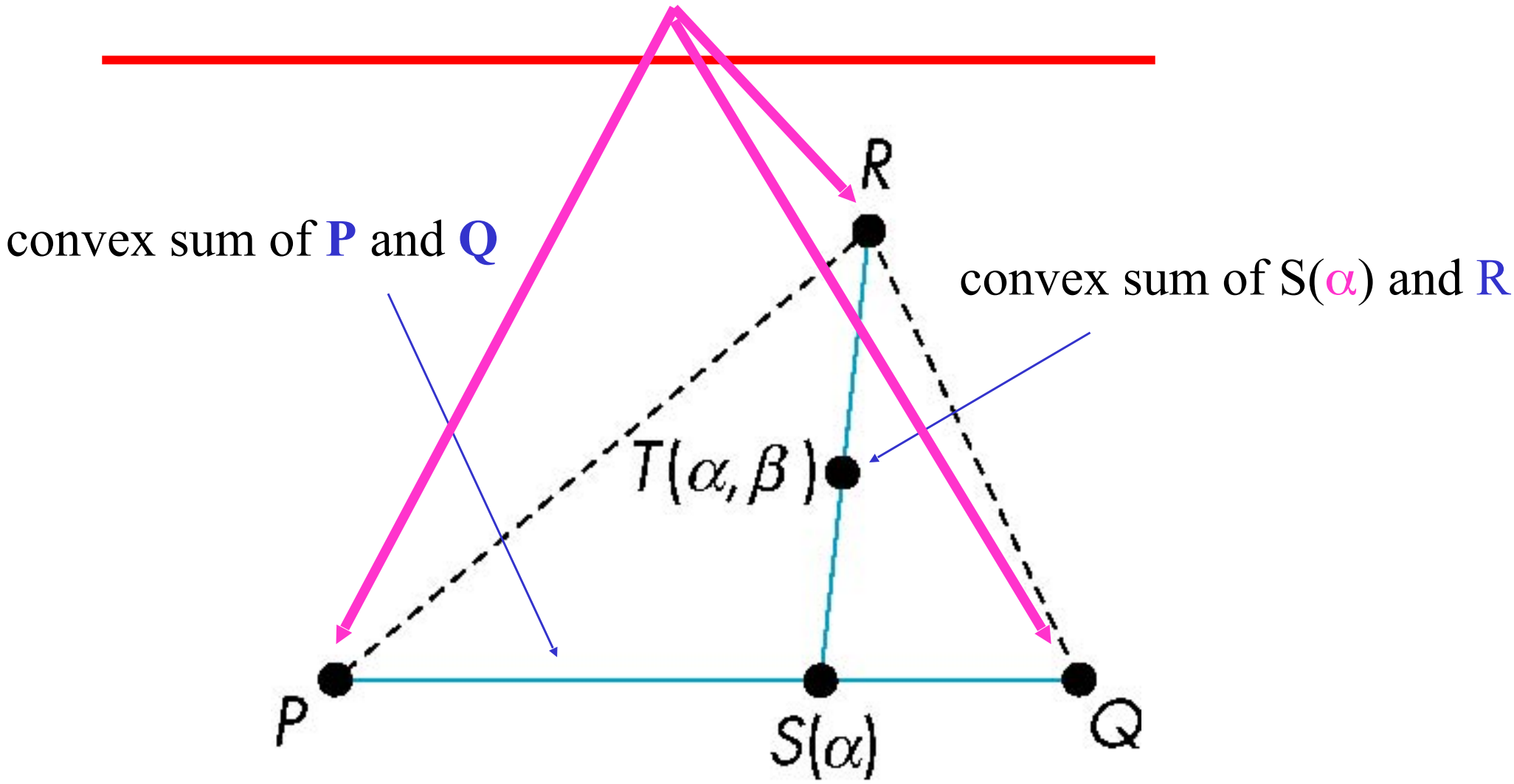




$P(u, v) = R + u\,\mathbf{a} + v\,\mathbf{b}$

$P(u, v) = R + u(Q - R) + v(P - R)$

Triangles



for 0<=α,β<=1, we get all points in triangle

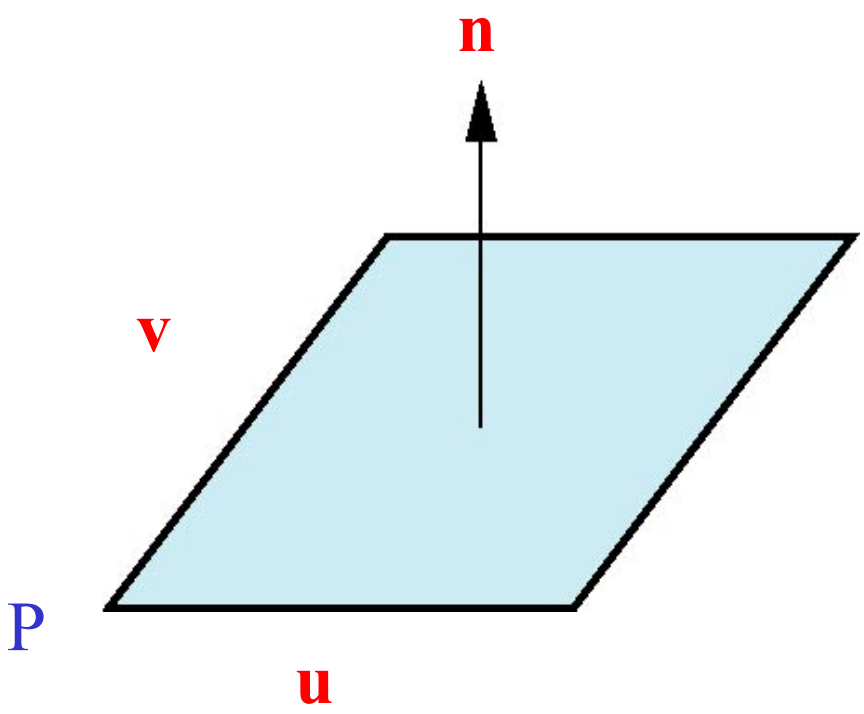
Normals

- Every **plane** has a vector **n** normal (**perpendicular,orthogonal**) to it

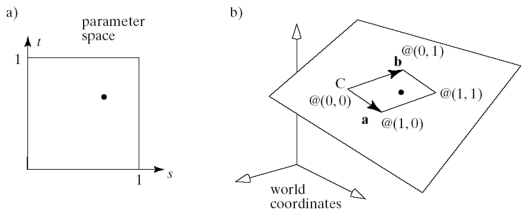
- From point-two vector form $P(\alpha,\beta)=P+\alpha\mathbf{u}+\beta\mathbf{v}$, we know we can use the **cross product** to find $\mathbf{n} = \mathbf{u} \times \mathbf{v}$

  and the equivalent form

  $(P(\alpha)-P) \bullet \mathbf{n} = 0$



Patches

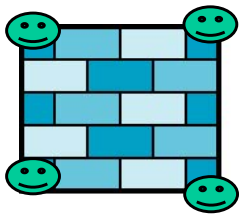


Mapping textures onto faces involves finding a **mapping** from a portion of parameter space onto **object space**.

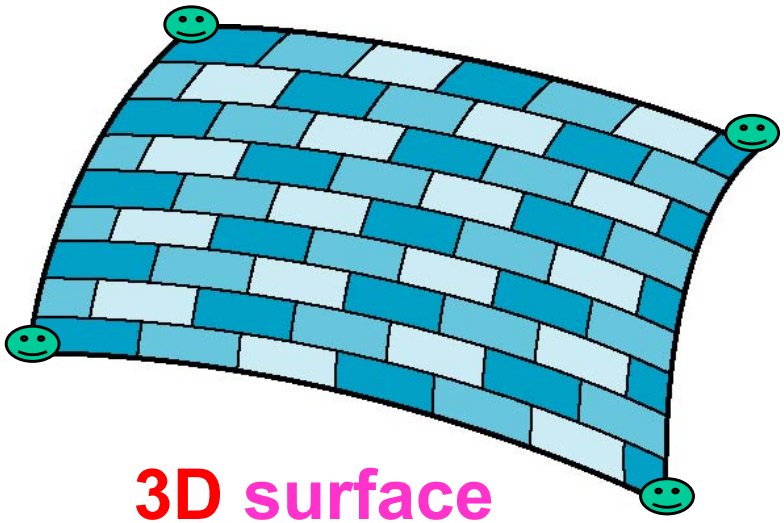
Each point (*s*, *t*) in parameter space corresponds to one **3D point in the patch** $P(s, t) = C + as + bt$.

The **patch is a parallelogram** whose corners correspond to the **four corners of parameter space** and are situated at:

$P(0, 0) = C$;
$P(1, 0) = C + a$;
$P(0, 1) = C + b$;
$P(1, 1) = C + a + b$.

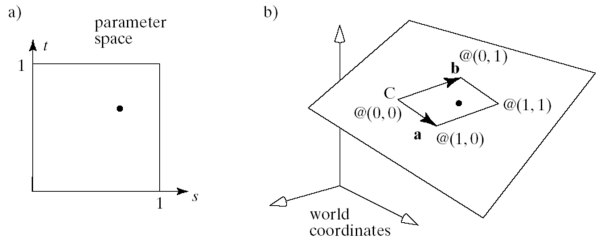


2D image



3D surface

Patches



The vectors **a** and **b** determine both the ***size and the orientation of the patch.***

If **a** and **b** are **perpendicular**, the ***patch will become rectangular.***

If in addition **a** and **b** have the **same length**, the ***patch will become square.***

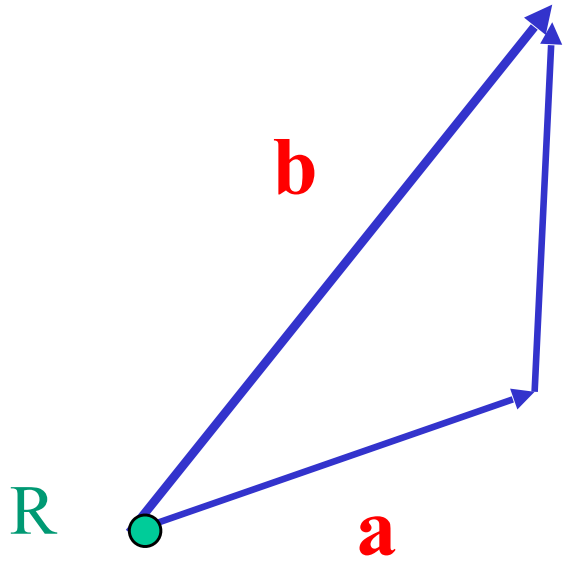
Changing *C* just ***translates the patch without changing its shape or orientation.***

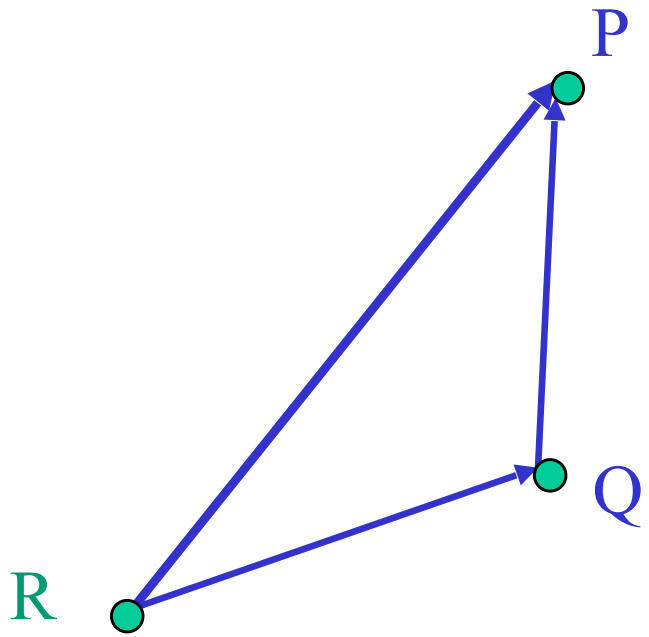
Planes – Parametric Form

A **plane** with two parameters u, v can be defined by

1 point R **and two vectors** (**a** and **b**) **or** by

3 points P, Q, R

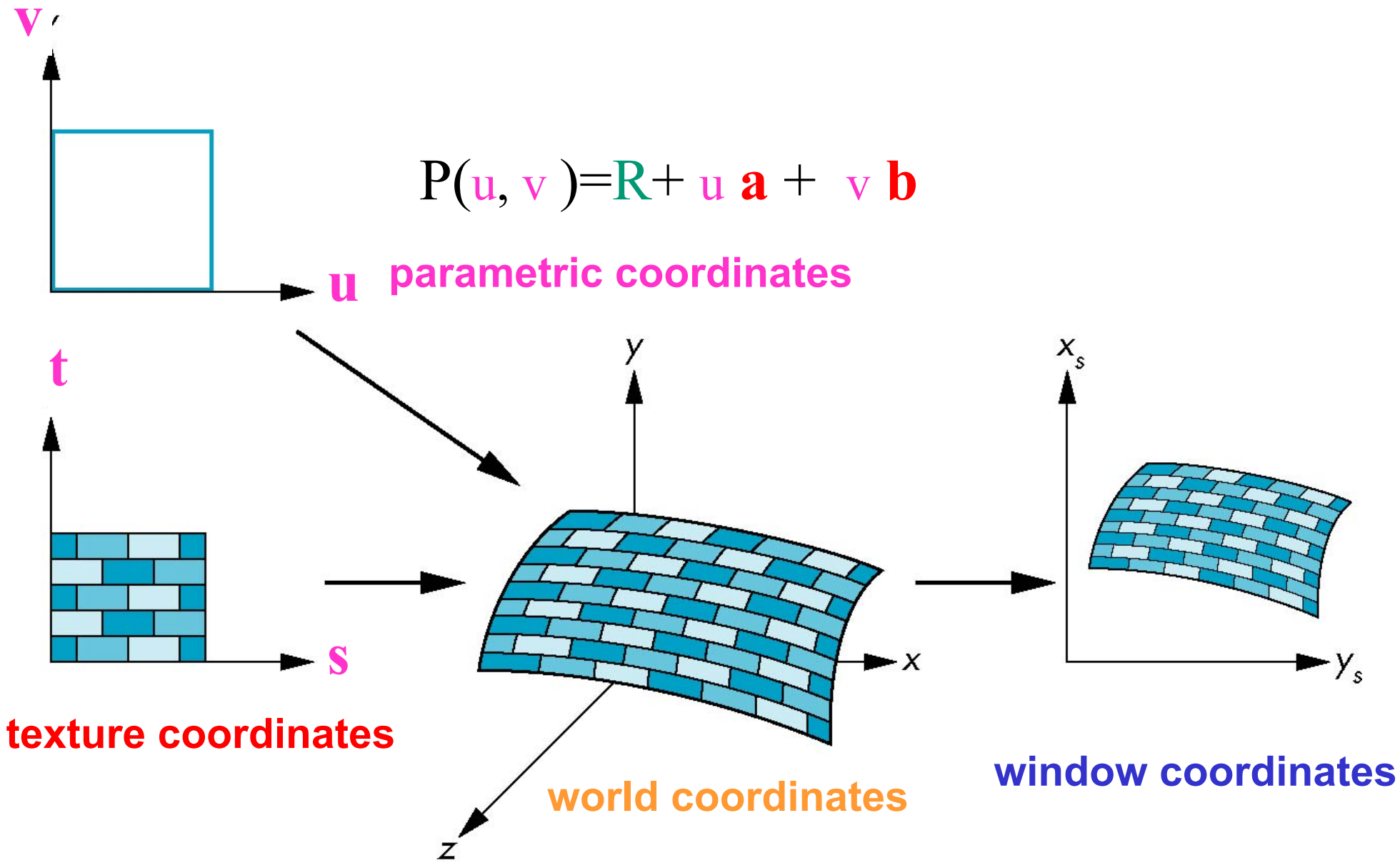




$P(u, v) = R + u\,\mathbf{a} + v\,\mathbf{b}$

$P(u, v) = R + u(Q - R) + v(P - R)$

Texture Mapping

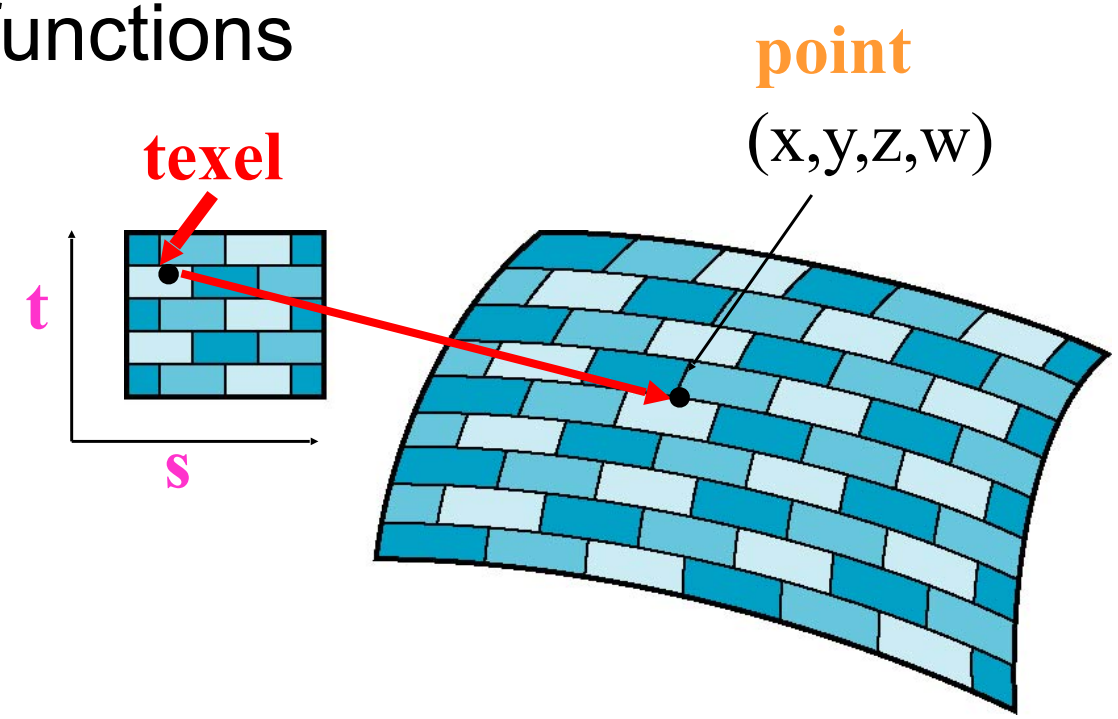


Mapping Functions

- **Basic problem is how to find the maps**
- Consider **mapping from texture coordinates to a point a surface**
- Appear to need three functions

$$x = x(s,t)$$
$$y = y(s,t)$$
$$z = z(s,t)$$
$$w = w(s,t)$$

??????



- **But we really want to go the other way**

Backward Mapping

We really want to go backwards

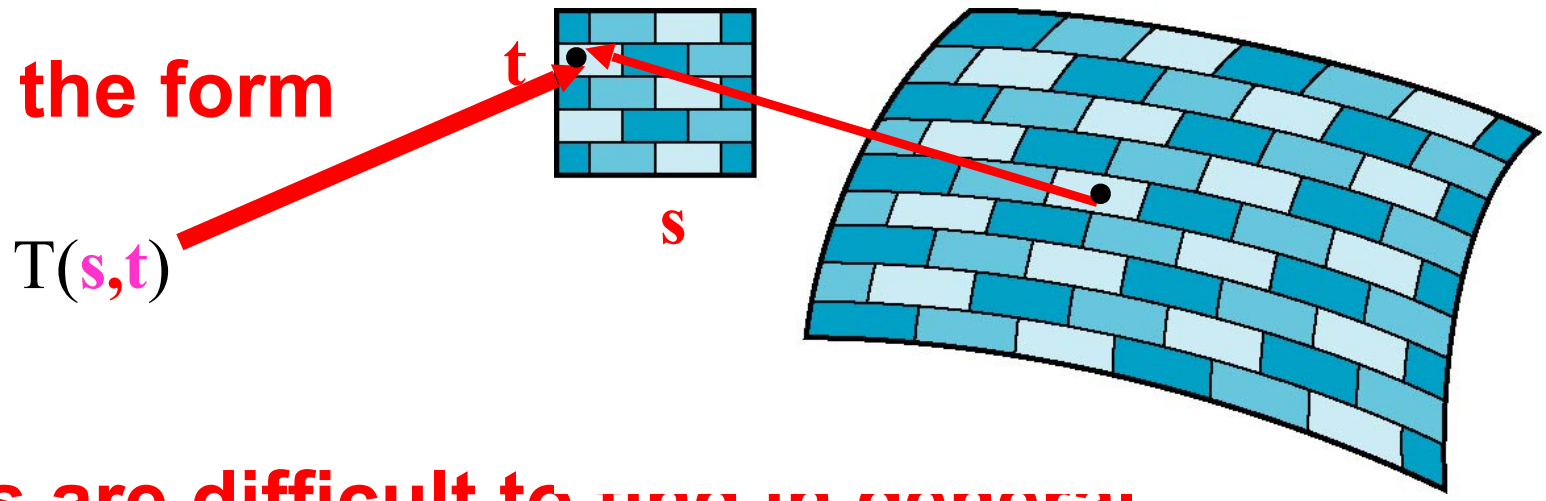
Given a **pixel**, we want to know to **which point on an object it corresponds**

Given a **point on an object**, we want to know to **which point in the texture it corresponds**

Need a map of the form

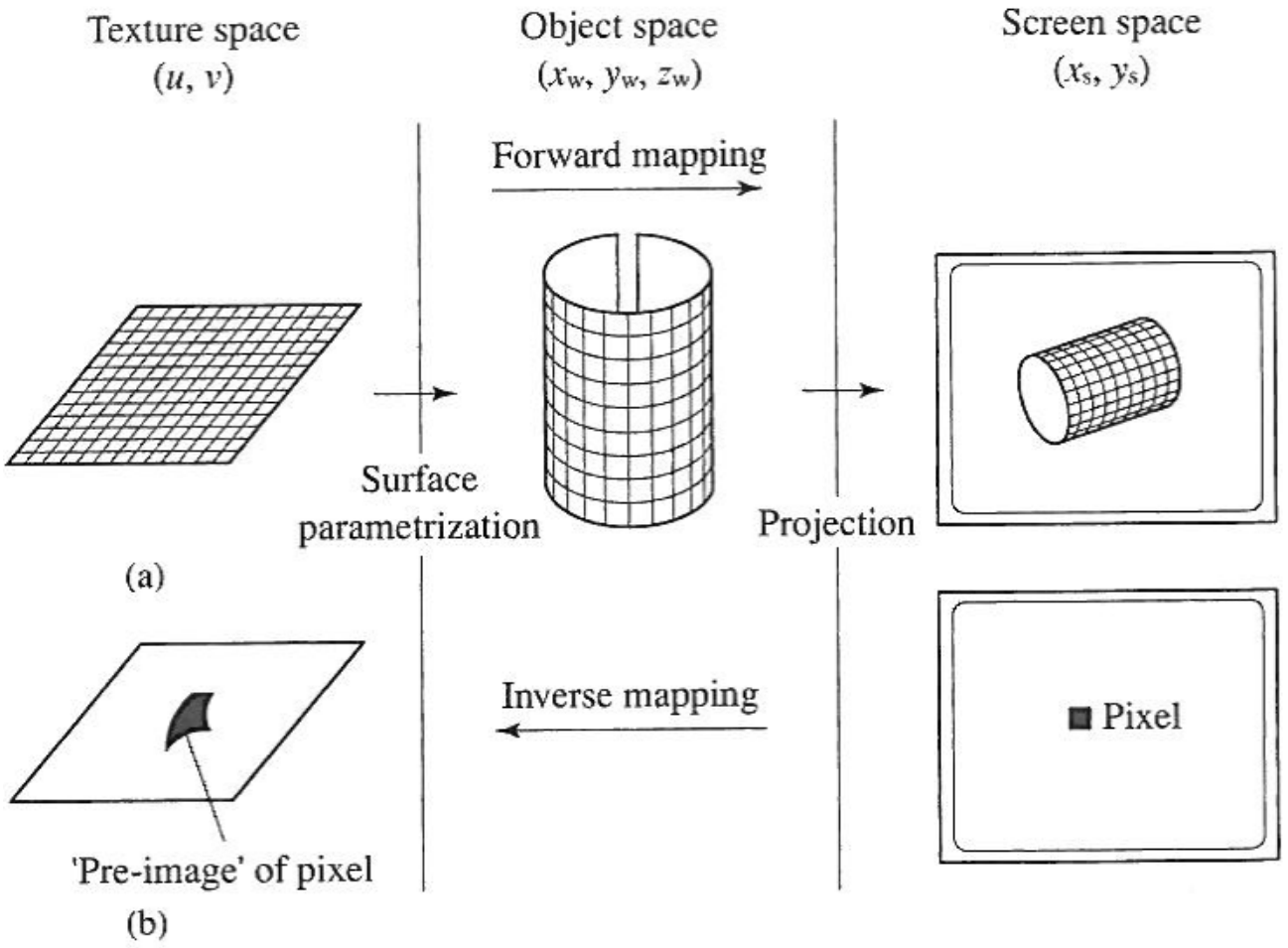
$s = s(x,y,z)$

$t = t(x,y,z)$

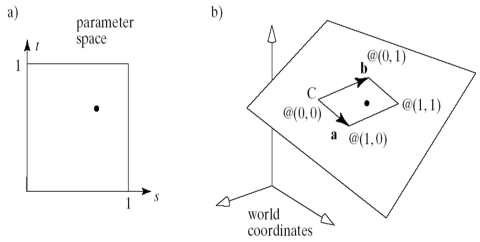


Such functions are difficult to find in general

Mappings – forward and inverse



Adding …Projection Process



Parametric form representation of a surface

$$P(u, v) = R + u(Q - R) + v(P - R)$$

define the **geometric object** using parametric (u, v) surfaces

additional **mapping** function that gives **object coordinate values**, (x, y, z) or (x, y, z,w) in terms of u and v

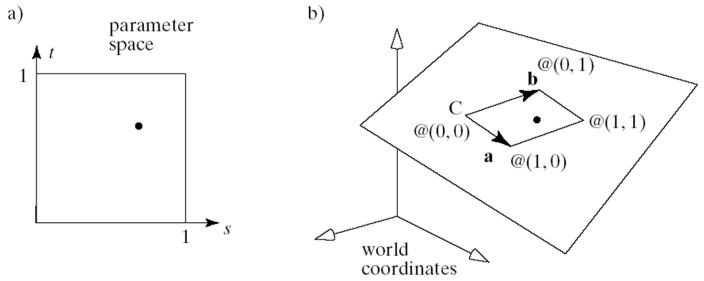
mapping from parametric coordinates (u, v) to **texture coordinates** s,t

projection process that take us from **object coordinates** **to** **window coordinates**, going through **eye coordinates**, **clip** coordinates, and **normalized device coordinates** along the way.

function that takes a **texture coordinate pair** (s, t) and tells us where in the **color buffe** the corresponding value of T(s, t) will make its contribution to the **final image**.

mapping of the form **xs** = xs(s, t), **ys** = ys(s, t). into window coordinates, where (xs , ys) is a location in the **color buffer**

Difficulties….



First, we must determine the **map** from **texture coordinates** to **object coordinates. A two-dimensional texture** usually is defined over a rectangular region in texture space. **The mapping** from **this rectangle** to **an arbitrary region in three-dimensional space** may be a **complex function** or may have undesirable properties. **For example, if we wish to map a rectangle to a sphere, we cannot do so without distortion of shapes and distances**.

Second, owing to the nature of the **rendering process**, which works on a pixel-by-pixel basis, we are more interested in the inverse **map** from **screen coordinates** to **texture coordinates**. It is when we are determining the shade of a **pixel** that we must determine **what point in the texture image** to use a calculation that requires us to go from **window coordinates** to **texture coordinates**.

Third, because each **pixel** corresponds to a small rectangle on the display, we are interested in **mapping** not points to points, but rather **areas** to **areas**. Here again is a potential **aliasing problem** that we must treat carefully if we are to avoid artifacts, such as **wavy sinusoidal** or **moire patterns**.

Mapping Problem

In computer graphics, most **curved surfaces** are represented parametrically **u,v**

A point **p** on the **surface** is a function of two parameters **u** and **v.**

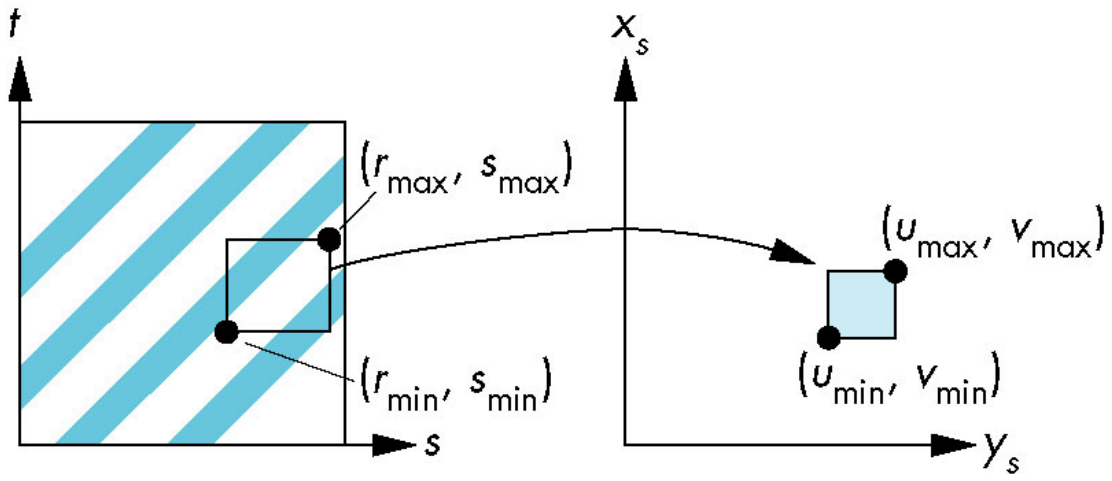
Given a **parametric surface**, we can often **map** a point in the **texture map T(s, t)** to a point on the surface **p(u, v)** by a **linear map** of the form u = a**s** + b**t** + c , v = d**s** + e**t** + f

As long as ae = bd, this **mapping** is invertible

Linear mapping makes it easy to **map** a **texture** to a group of **parametric surface patches**.

This mapping is easy to apply, but it does not take into account the curvature of the surface.

Equal-sized texture patches must be stretched to fit over the surface patch.

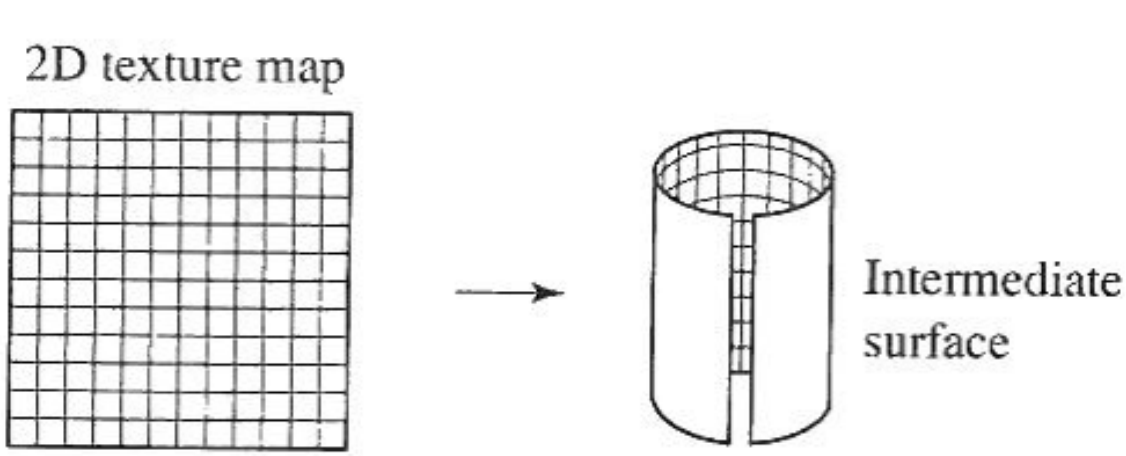


Two-part mapping

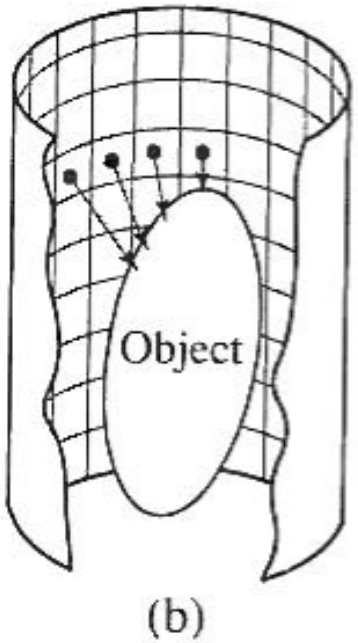
First map the texture to simple intermediate surface

- cylinder
- sphere
- box

Two-stage mapping as a forward process



$$T(u, v) \rightarrow T'(x_i, y_i, z_i)$$



(a) **S** mapping (b) **O** mapping

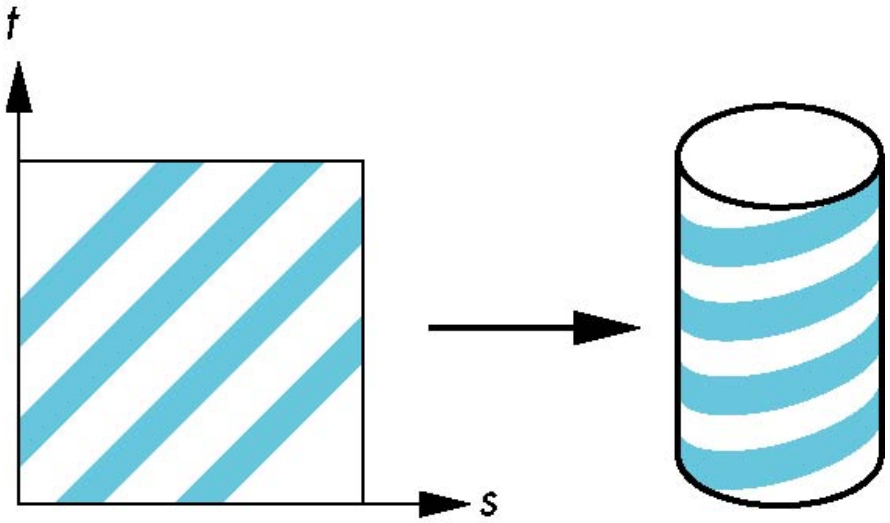
$$T'(x_i, y_i, z_i) \rightarrow O(x_w, y_w, z_w)$$

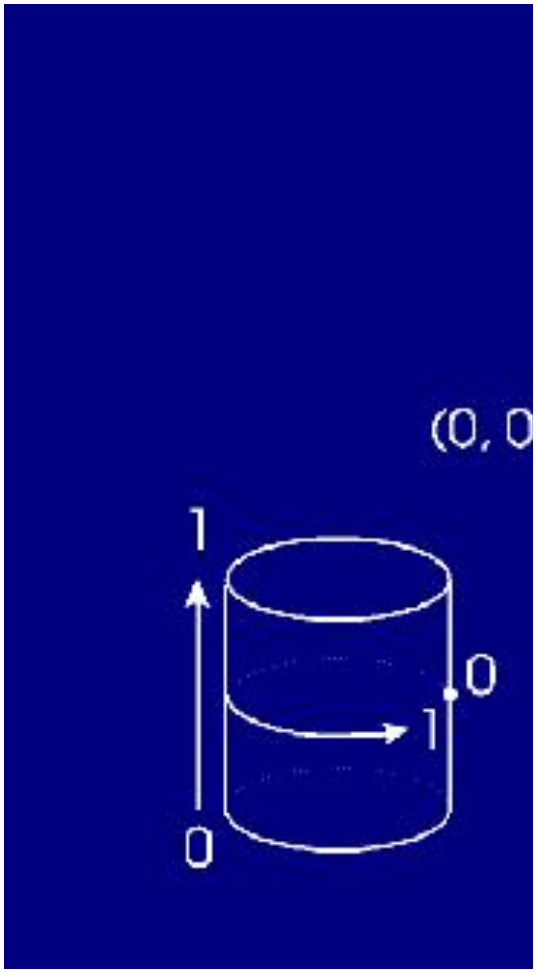
$$S_{\text{cylinder}}: (\theta, h) \rightarrow (u, v)$$

$$= \left(\frac{r}{c} (\theta - \theta_0), \frac{1}{d} (h - h_0) \right)$$

First-part mapping T to T’

map to cylinder





Cylindrical Mapping

Suppose texture coordinates vary over the unit square

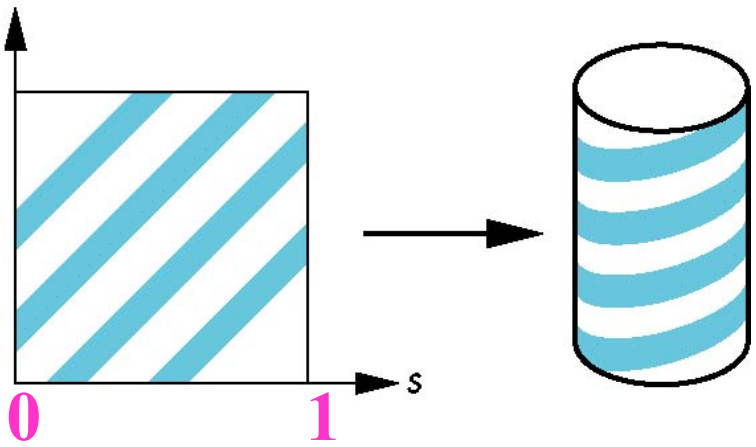
Points (x,y,z) on the cylinder are given by the parametric equations of the cylinder

$$x = r \cos 2\pi u$$
$$y = r \sin 2\pi u$$
$$z = v/h$$

maps rectangle in u,v space to cylinder of radius r and height h in world coordin

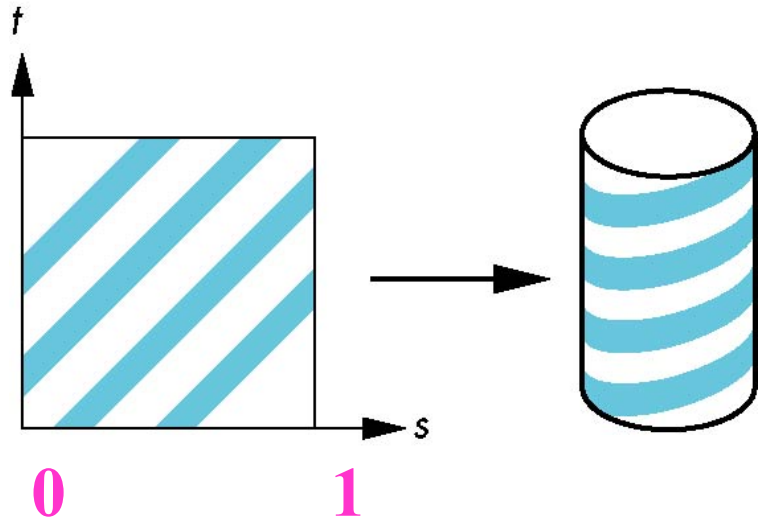
$$s = u$$
$$t = v$$

maps from texture space



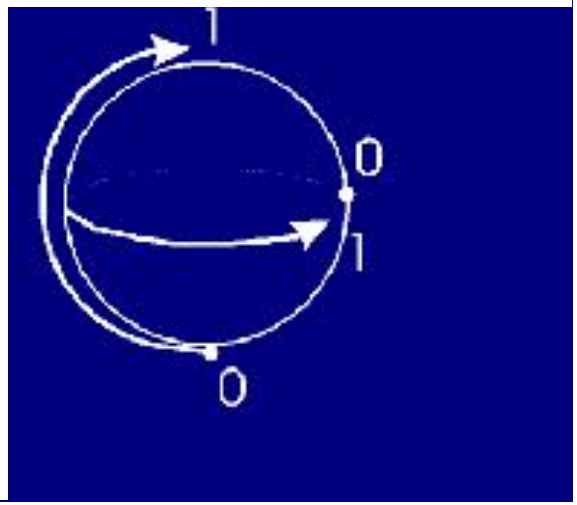
Cylindrical Mapping

By using only the curved part of the cylinder, and not the top and bottom, we were able to map the texture without distorting its shape.



First-part mapping

map to sphere



Spherical Map

We can use a **parametric sphere**

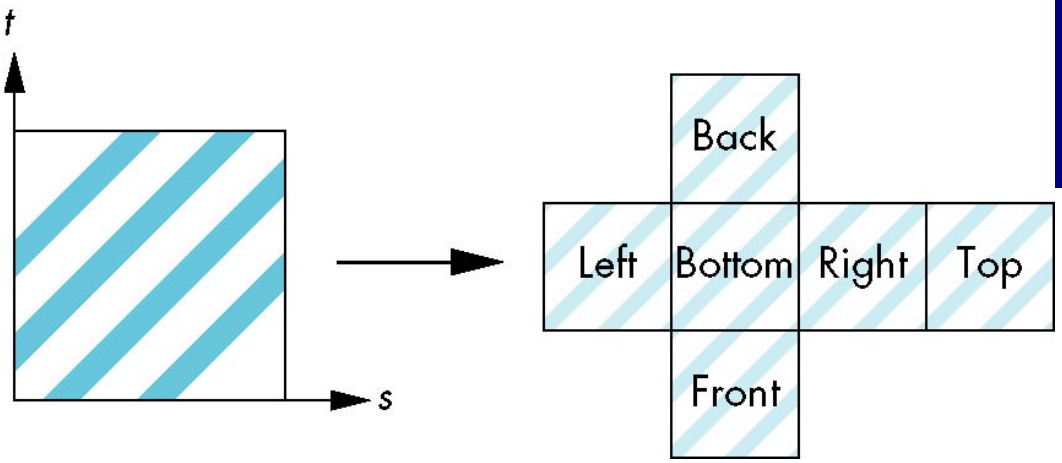
$$x = r \cos 2\pi u$$
$$y = r \sin 2\pi u \cos 2\pi v$$
$$z = r \sin 2\pi u \sin 2\pi v$$

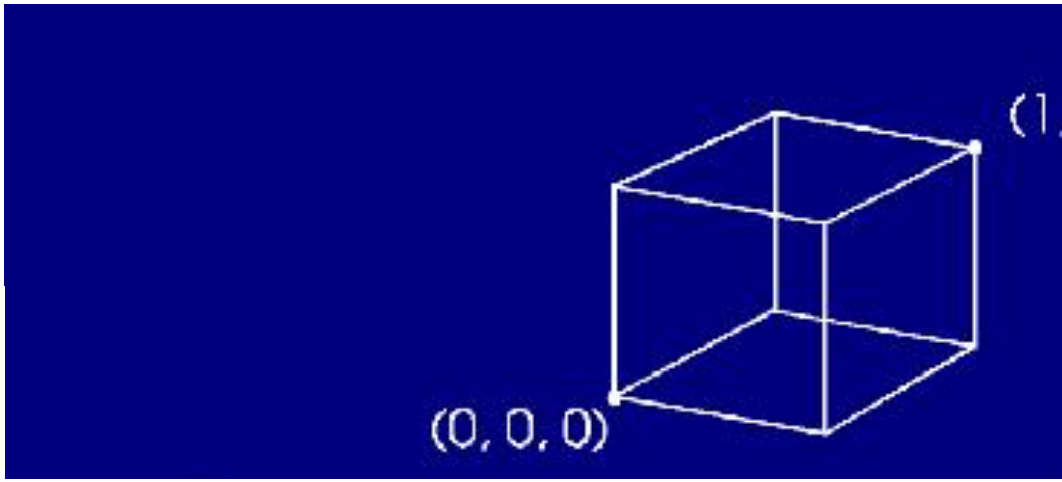
in a similar manner to the **cylinder** but have to decide where to put the distortion (for example, the familiar Mercator projection puts the most distortion at the poles)

Spheres are used in environmental maps

First-part mapping

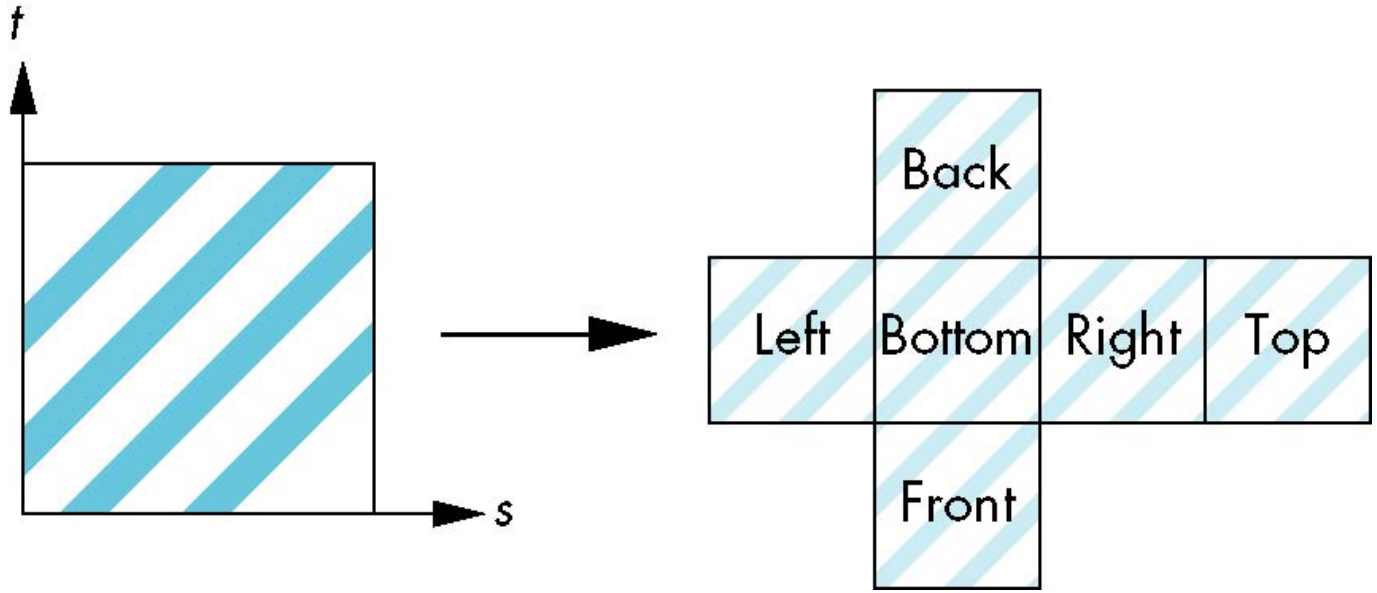
map to box



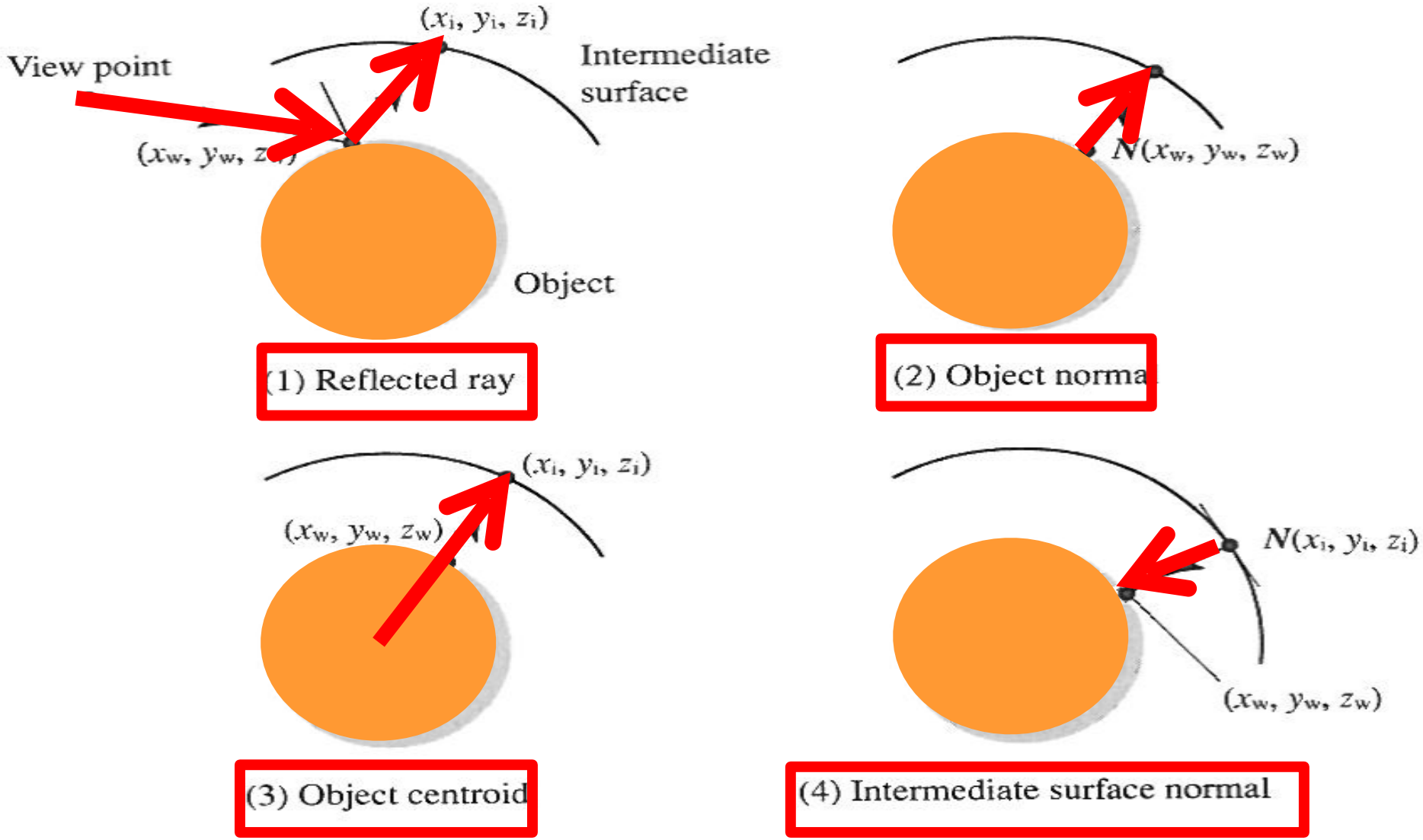


Box Mapping

- Easy to use with **simple orthographic projection**
- Here we **map** the **texture** to a **box** that can be unraveled, like a cardboard packing box
- Also used in **environment maps**

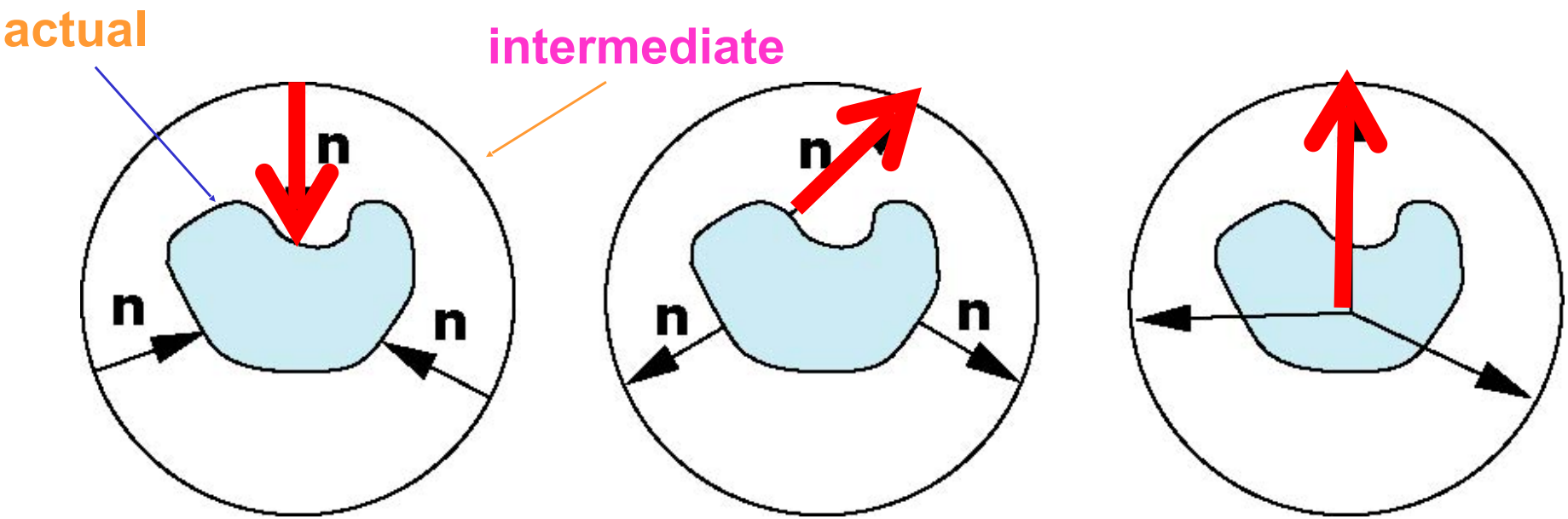


Second Mapping – Intermediate Surface T' onto Object O



Second Mapping

- Map from **intermediate object** to **desired surface**
  - **Normals** from **intermediate** to **actual**
  - **Normals** from **actual** to **intermediate**
  - **Vectors** from center of **intermediate**

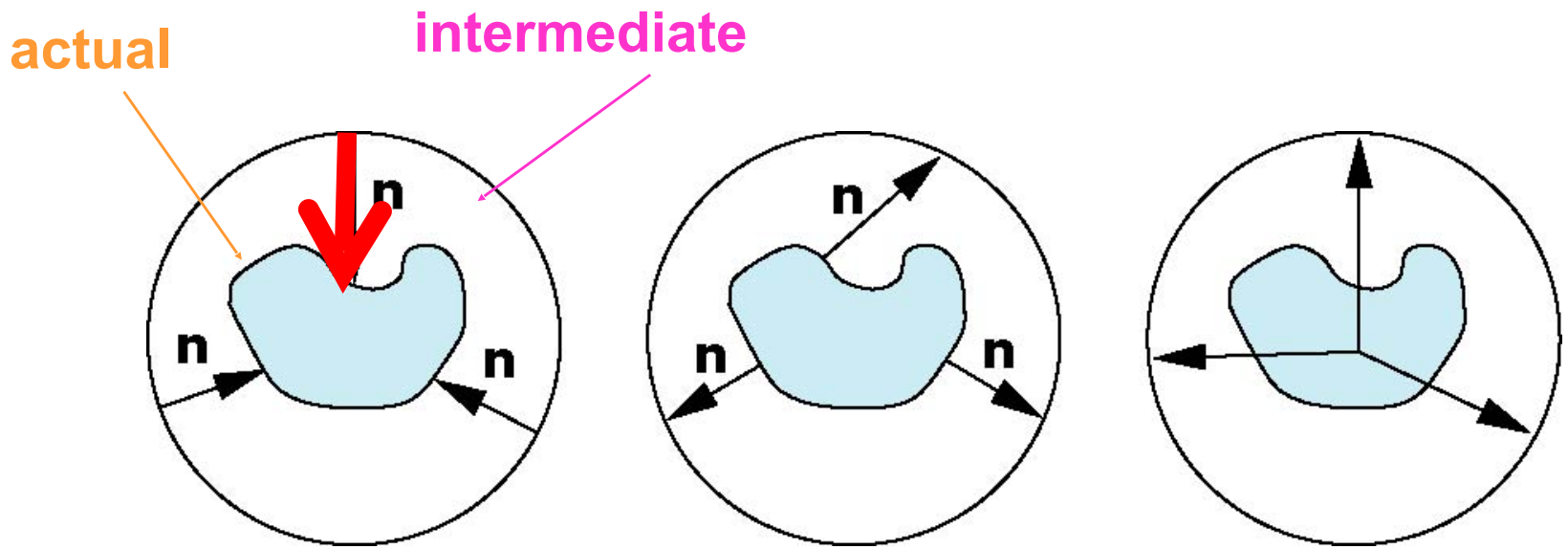


Second Mapping - 1

Map from **intermediate object** to **desired surface**

Normals from **intermediate** to **actual**

take the **texture value at a point on the intermediate object**, go from this point in the direction of the normal until we **intersect the object**, and then place the texture value at the **point of intersection**.

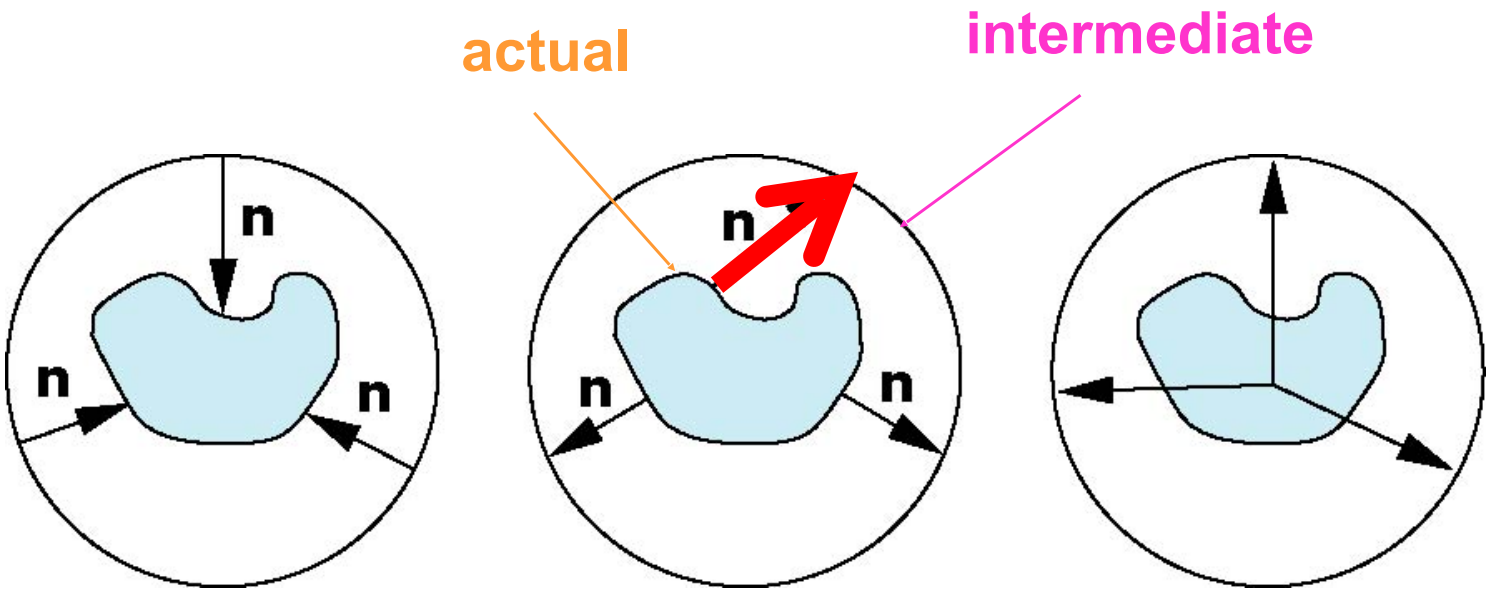


Second Mapping - 2

- Map from **intermediate object** to **desired surface**

  Normals from **actual** to **intermediate**

  starting at a **point on the surface of the object** and going in the direction of the **normal** at this point until we **intersect the intermediate object**, where we obtain the texture value

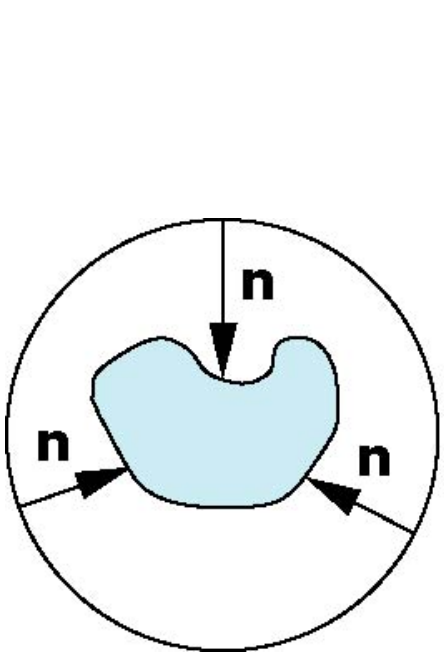


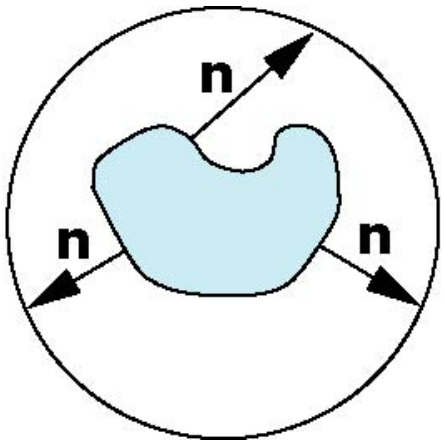
Second Mapping - 3

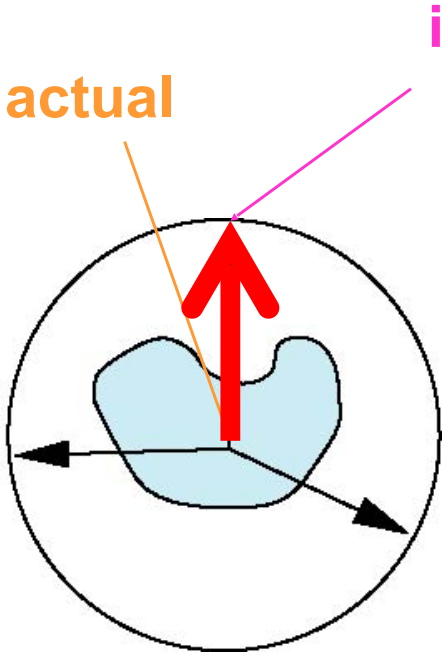
- Map from **intermediate object** to **desired surface**

  Vectors from **center of intermediate**

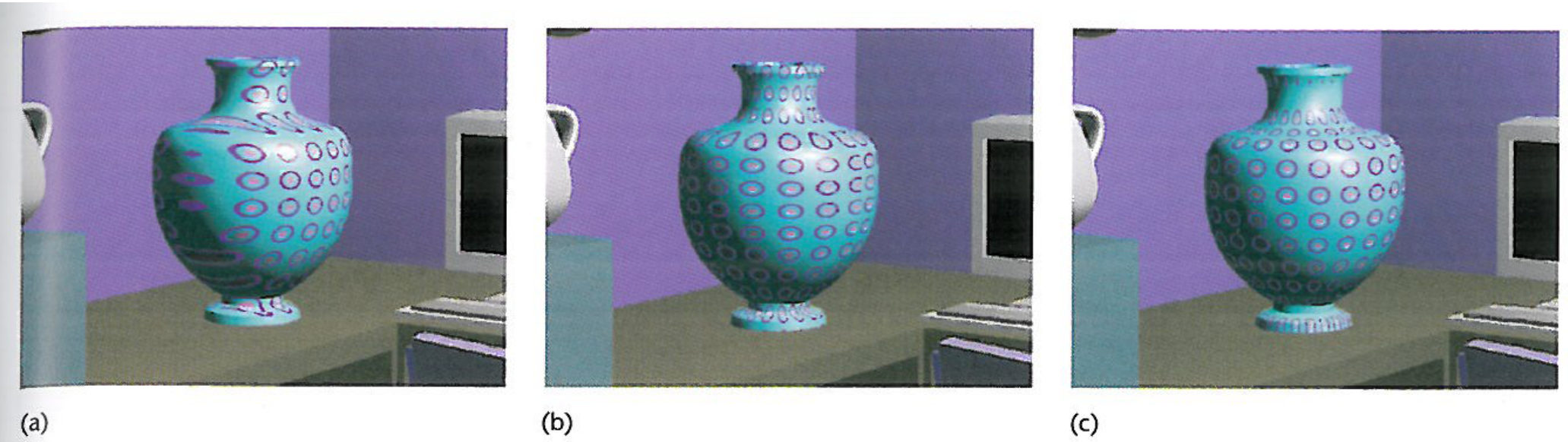
  if we know the **center of the object**, is to draw a line from the **center through a point on the object**, and to **calculate the intersection** of this line with the **intermediate surface**







Example of the three texture mappings



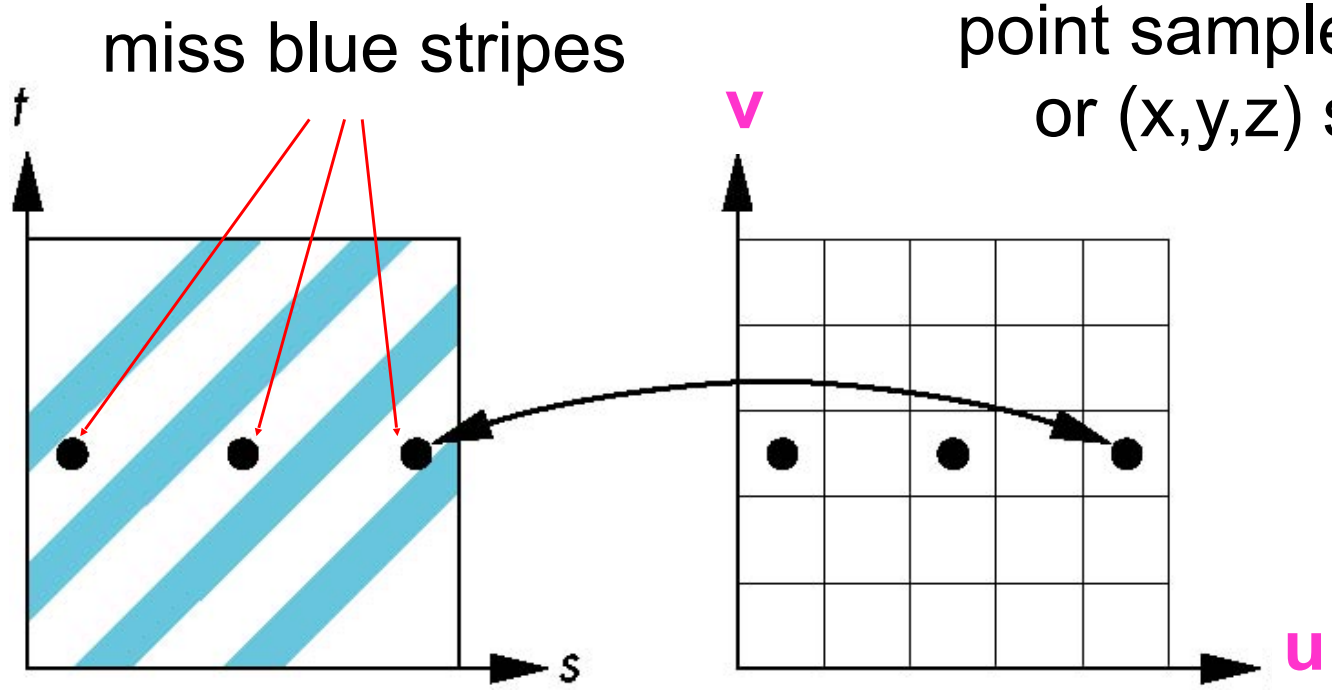
(a) (b) (c)

Figure 8.7
Examples of two-part texture mapping with a solid of revolution. The intermediate surfaces are: (a) a plane (or no surface); (b) a cylinder; and (c) a sphere.

A Plane A Cylinder A sphere

Aliasing

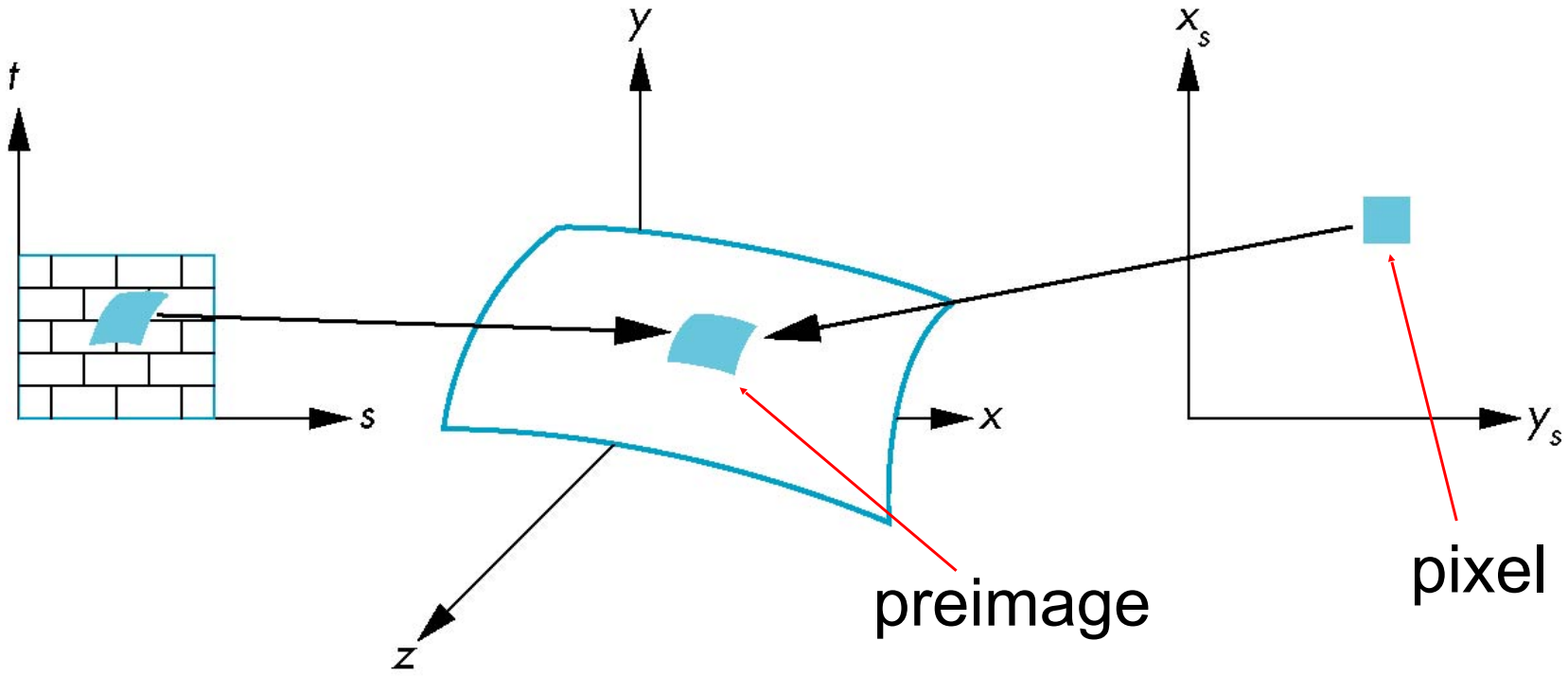
- **Point sampling** of **the texture** can lead to **aliasing errors**



point samples in **texture space (s,t)**

Area Averaging

A better but slower option is to use ***area averaging***



Note that *preimage* of **pixel** is curved

OpenGL Texture Mapping

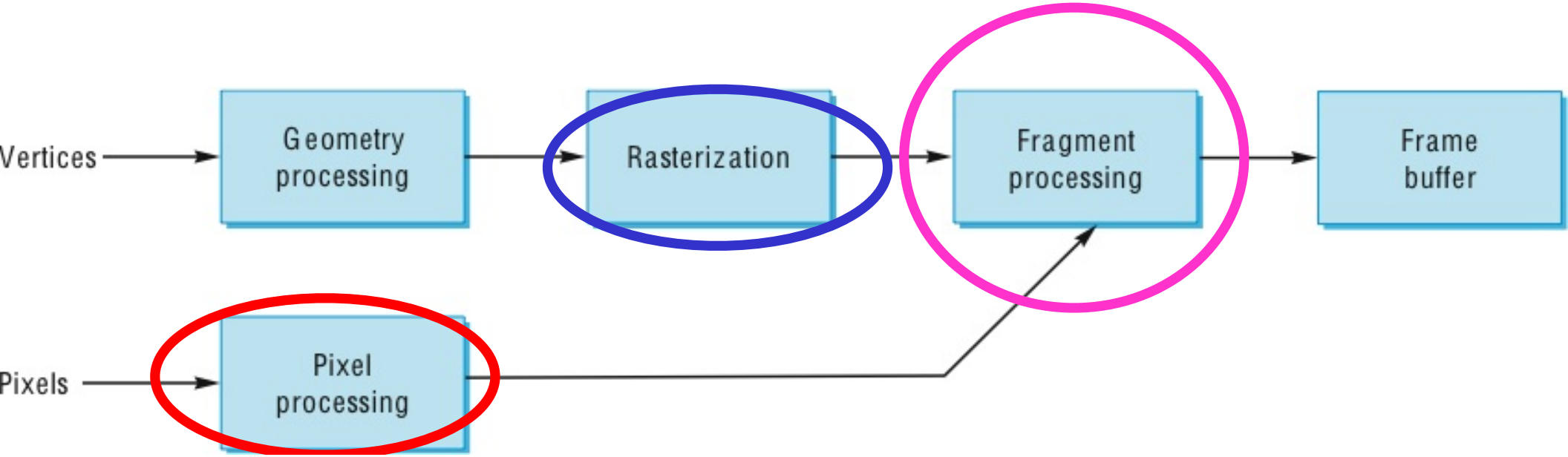
Chapter 8.8

Objectives

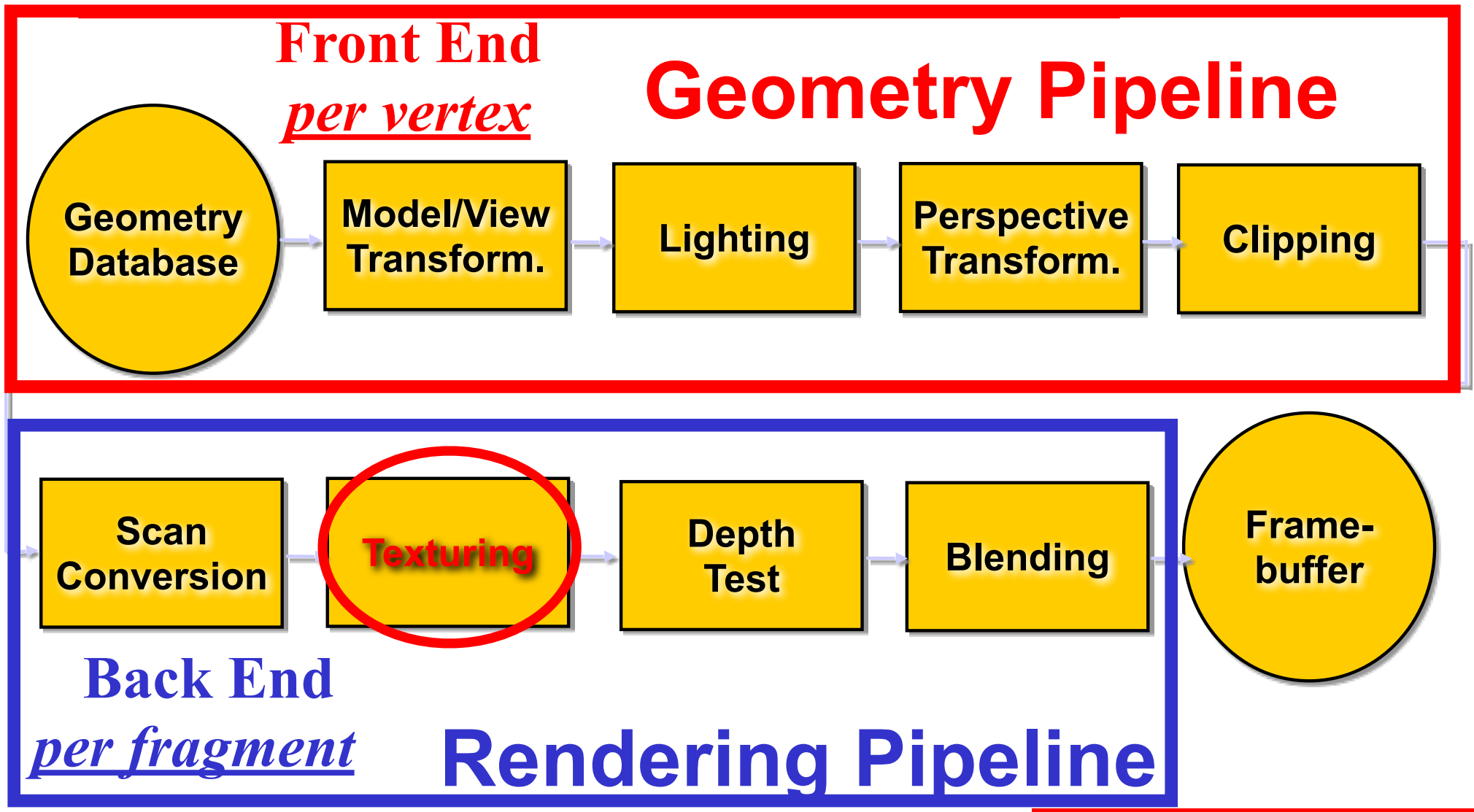
Introduce the OpenGL texture functions and options

We focus on mapping two-dimensional textures to surfaces

Texture coordinates are handled much like **normals** and **colors**. **They are associated** with vertices through the **OpenGL state** and the required **texture values** can then be obtained by **interpolating the texture coordinates** at the vertices **across** polygons.



The Graphics Pipeline

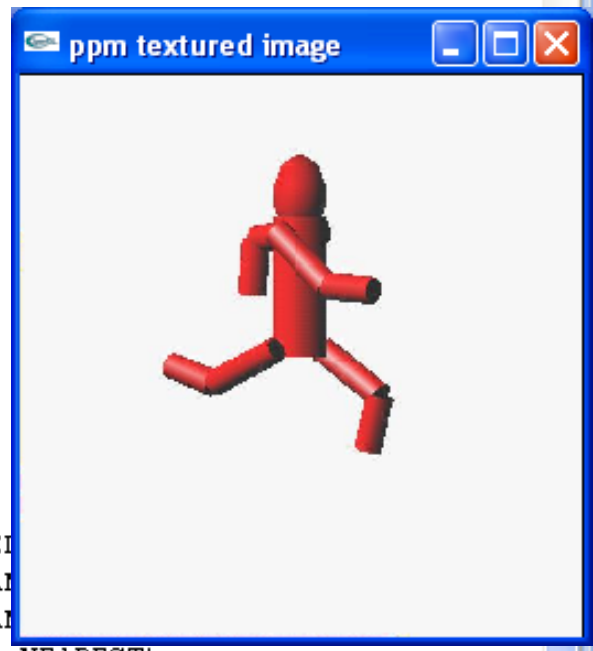


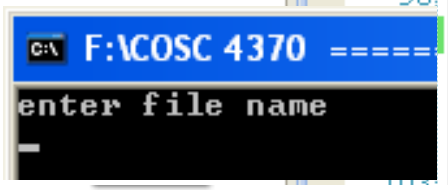
CLASS PARTICIPATION 4!
(Next Slide)

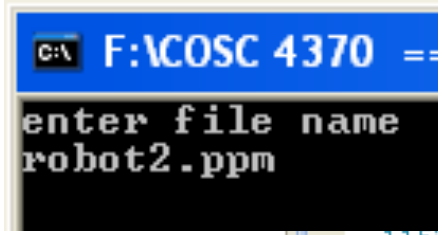
ppmtex.c

```
        image[3*nm-3*i-2]=green;
        image[3*nm-3*i-1]=blue;
    }
    printf("read image\n");
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
    glutInitWindowSize(n, m);
    glutInitWindowPosition(0,0);
    glutCreateWindow("ppm textured image");
    glutReshapeFunc(myreshape);
    glutDisplayFunc(display);
    glPixelTransferf(GL_RED_SCALE, s);
    glPixelTransferf(GL_GREEN_SCALE, s);
    glPixelTransferf(GL_BLUE_SCALE, s);
    glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
    glPixelStorei(GL_UNPACK_ALIGNMENT,1);
    glEnable(GL_TEXTURE_2D);
    glTexImage2D(GL_TEXTURE_2D,0,3,n,m,0,GL_RGB,GL_UNSIGNE
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLA
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLA
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0,1.0,1.0);
    glutMainLoop();


}
```







Basic Strategy

Three steps to applying a texture

1. **specify the texture**
   - read or generate image
   - assign to texture
   - enable texturing

```
glTexImage2D(GL_TEXTURE_2D,0,3,n,m,0,GL_RGB,
             GL_UNSIGNED_INT, image);
```

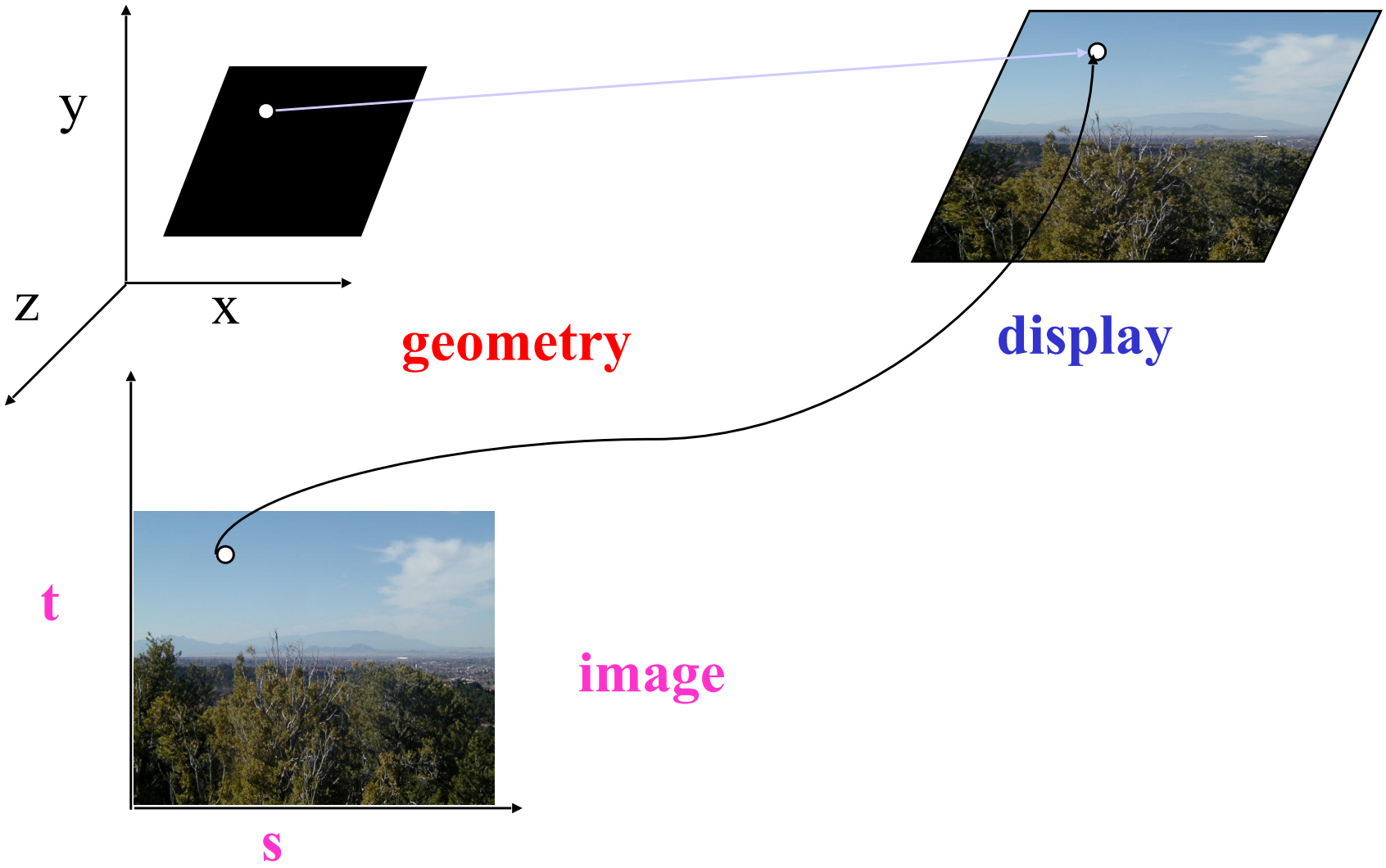
```
glEnable(GL_TEXTURE_2D);
```

2. **assign texture coordinates to vertices**
   - Proper mapping function is left to application

3. **specify texture parameters**
   - wrapping, filtering

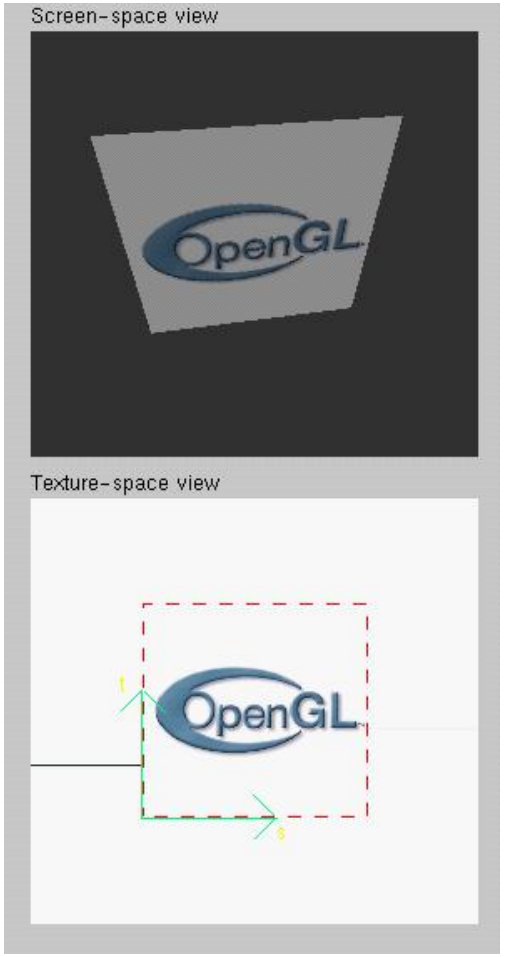
```
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,  GL_CLAMP);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,  GL_CLAMP);
```

Texture Mapping



Texture Example

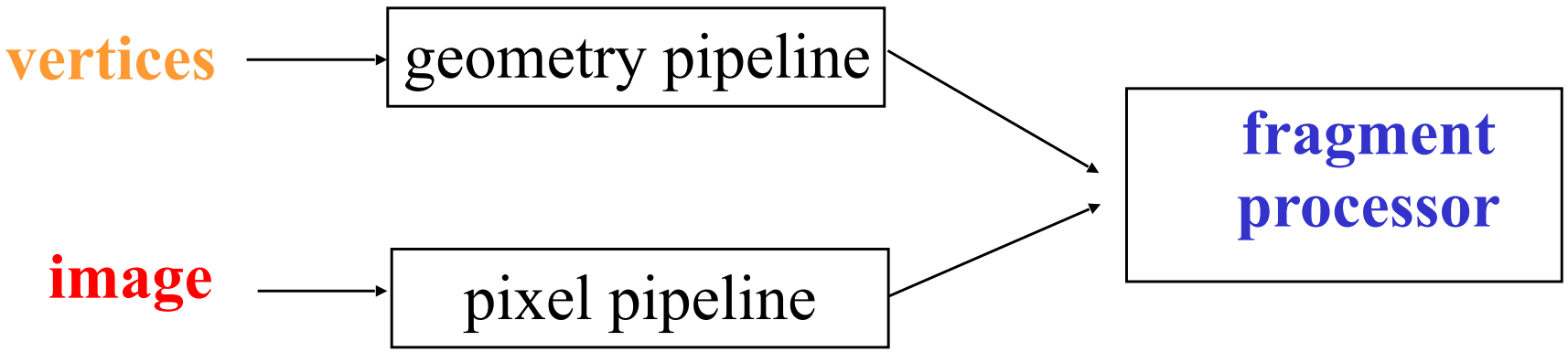
- The **texture** (below) is a **256 x 256 Image** that has been **mapped** to a **rectangular polygon** which is **viewed in perspective**



Texture Mapping and the OpenGL Pipeline

geometry and Images flow through separate pipelines that join during fragment processing

"complex" textures do not affect geometric complexity



Specifying a Texture Image

- Define a texture image from an array of *texels* (texture elements) in **CPU memory**

```
Glubyte my_texels[512][512];
```

- Define as any other pixel map
  - Scanned image
  - Generate by application code

- Enable texture mapping

```
glEnable(GL_TEXTURE_2D)
```

OpenGL supports 1-4 dimensional texture maps

8 - Microsoft Visual Studio

View Project Build Debug Tools Visual Assert Test Window Help

Debug | Win32 | rasterpos

rer - Solution 'Lecture ...

'Lecture 8' (1 project)
ture 8
Header Files
Resource Files
Source Files
ppmtex.c

ppmtex.c

(Global Scope) | main(int argc, char ** argv)

```
91          image[3*nm-3*i-2]=green;
92          image[3*nm-3*i-1]=blue;
93      }
94      printf("read image\n");
95      glutInit(&argc, argv);
96      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
97      glutInitWindowSize(n, m);
98      glutInitWindowPosition(0,0);
99      glutCreateWindow("ppm textured image");
100     glutReshapeFunc(myreshape);
101     glutDisplayFunc(display);
102     glPixelTransferf(GL_RED_SCALE, s);
103     glPixelTransferf(GL_GREEN_SCALE, s);
104     glPixelTransferf(GL_BLUE_SCALE, s);
105     glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
106     glPixelStorei(GL_UNPACK_ALIGNMENT,1);
107     glEnable(GL_TEXTURE_2D);
```

glEnable(GL_TEXTURE_2D)

```
GB,GL_UNSIGNED_INT, image);
WRAP_S,GL_CLAMP);
110     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
111     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
112     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
113     glClearColor(0.0, 0.0, 0.0, 1.0);
114     glColor3f(1.0,1.0,1.0);
115     glutMainLoop();
116
117
118 }
119
```

Define Image as a Texture

```
glTexImage2D(target,level,components,w,h,border,format,type,texels)
```

`target:` type of texture, e.g. `GL_TEXTURE_2D`

`level:` used for mipmapping (discussed later)

`components:` elements per texel

`w, h:` width and height of `texels` in pixels

`border:` used for smoothing (discussed later)

`format and type:` describe texels

`texels:` pointer to texel array

```
glTexImage2D(GL_TEXTURE_2D,0,3,512,512,0,GL_RGB,GL_UNSIGNED_BYTE,my_texels);
```

Debug Tools Visual Assert Test Window Help

Debug | Win32 | rasterpos

ppmtex.c

(Global Scope) | main(int argc, char ** argv)

```
91          image[3*nm-3*i-2]=green;
92          image[3*nm-3*i-1]=blue;
93      }
94      printf("read image\n");
95      glutInit(&argc, argv);
96      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
97      glutInitWindowSize(n, m);
98      glutInitWindowPosition(0,0);
99      glutCreateWindow("ppm textured image");
100     glutReshapeFunc(myreshape);
101     glutDisplayFunc(display);
102     glPixelTransferf(GL_RED_SCALE, s);
103     glPixelTransferf(GL_GREEN_SCALE, s);
104     glPixelTransferf(GL_BLUE_SCALE, s);
105     glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
106     glPixelStorei(GL_UNPACK_ALIGNMENT,1);
107     glEnable(GL_TEXTURE_2D);
108     glTexImage2D(GL_TEXTURE_2D,0,3,n,m,0,GL_RGB,GL_UNSIGNED_INT, image);
109     glTexImage2D(GL_TEXTURE_2D,0,3,512,512,0,GL_RGB,GL_UNSIGNED_BYTE,my_texels);
110     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
111     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
112     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
113     glClearColor(0.0, 0.0, 0.0, 1.0);
114     glColor3f(1.0,1.0,1.0);
115     glutMainLoop();
116
117
```

Converting A Texture Image

OpenGL requires texture dimensions to be powers of 2

If dimensions of **image** are not powers of 2

```
gluScaleImage(format, w_in, h_in,
              type_in, *data_in, w_out, h_out,
              type_out, *data_out );
```

`data_in` is source image

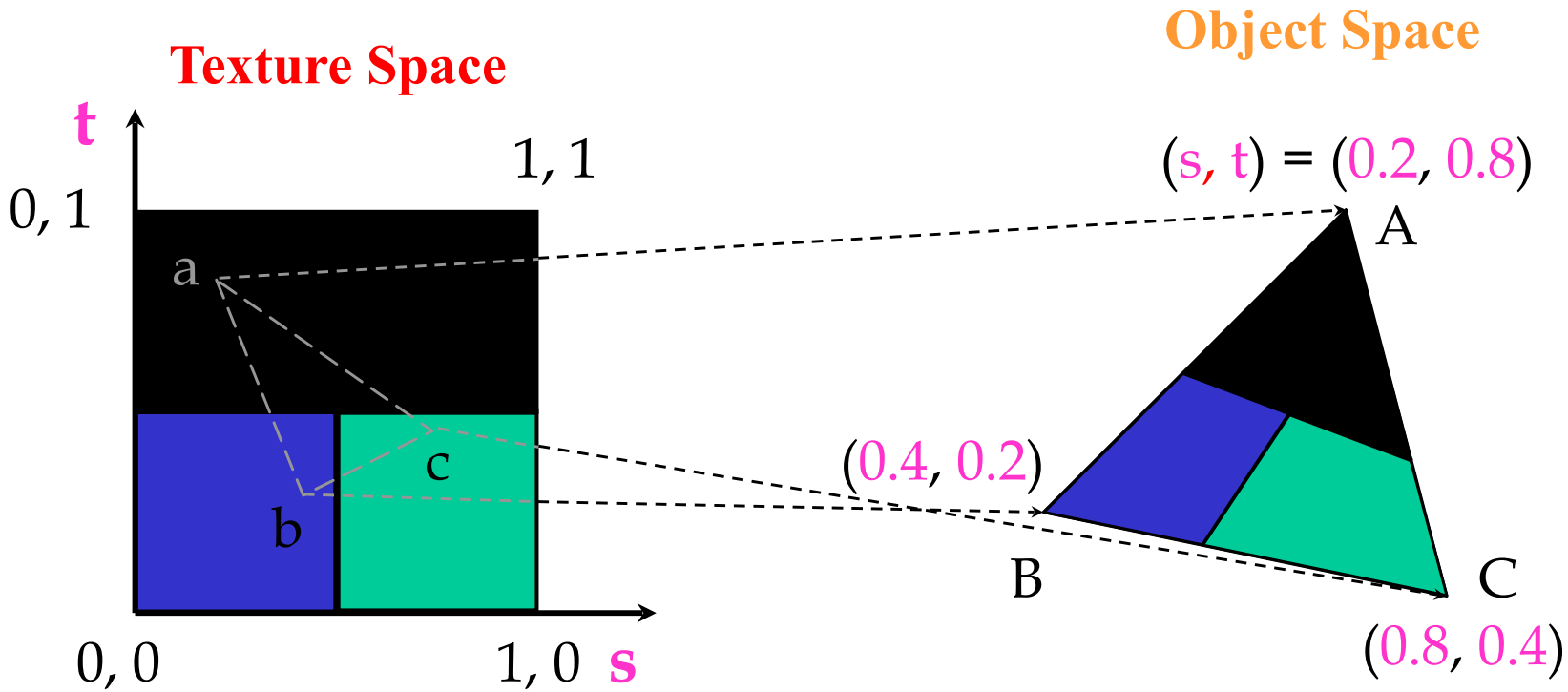
`data_out` is for destination image

Image interpolated and filtered during scaling

Mapping a Texture

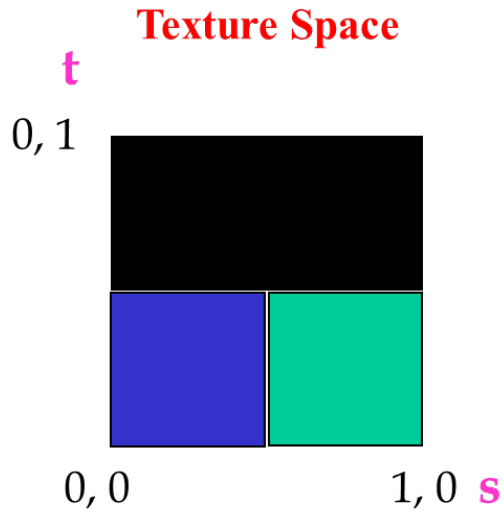
- Based on **parametric texture coordinates**

`glTexCoord*()` specified at **each vertex**

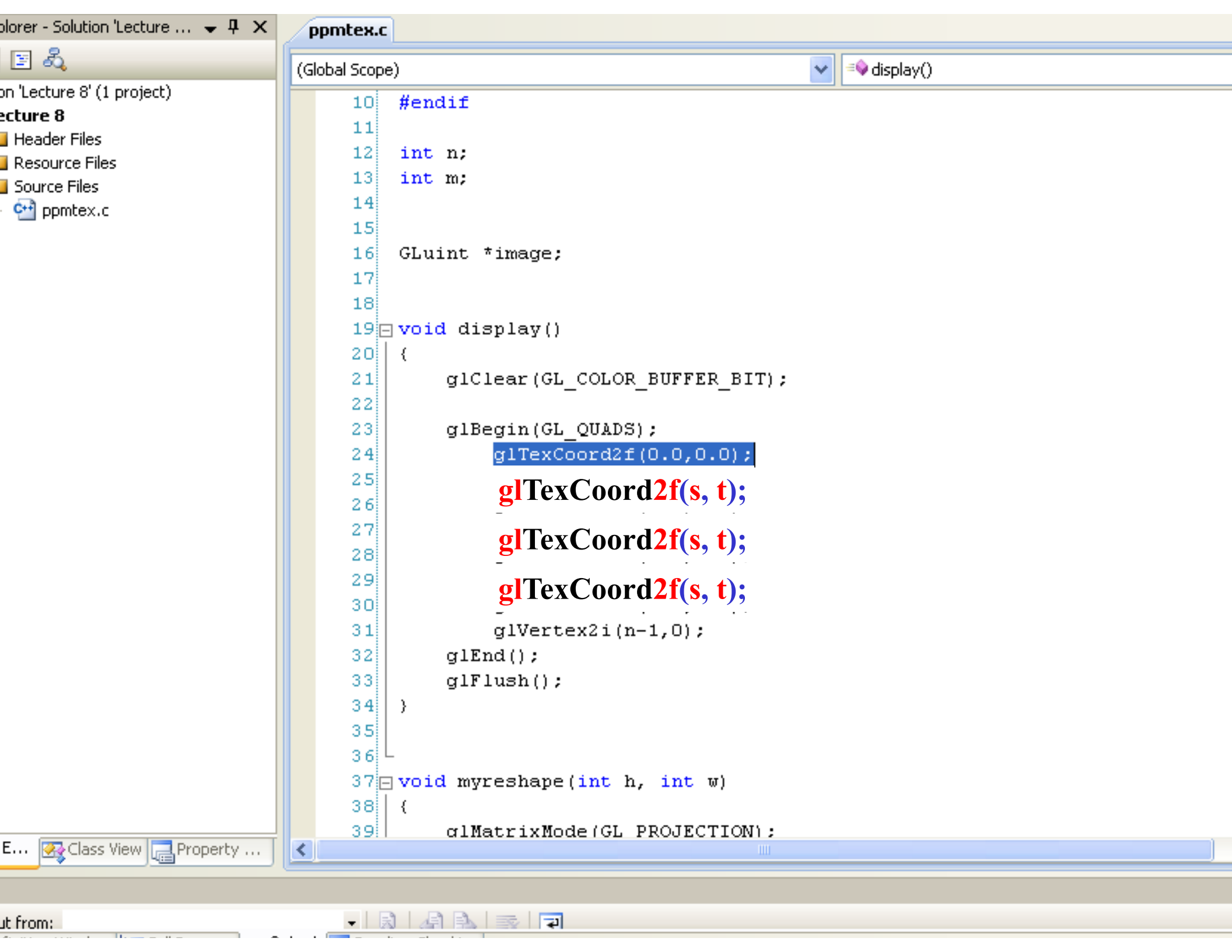


Typical Code

```
glBegin(GL_POLYGON);
  glColor3f(r0, g0, b0);   //if no shading used
  glNormal3f(u0, v0, w0); // if shading used
  glTexCoord2f(s0, t0);
  glVertex3f(x0, y0, z0);
  glColor3f(r1, g1, b1);
  glNormal3f(u1, v1, w1);
  glTexCoord2f(s1, t1);
  glVertex3f(x1, y1, z1);
        .
        .
glEnd();
```



Note that we can use vertex arrays to increase efficiency


olorer - Solution 'Lecture ...
on 'Lecture 8' (1 project)
ecture 8
Header Files
Resource Files
Source Files
ppmtex.c
E...
Class View
Property ...
ppmtex.c
(Global Scope)
display()
10 #endif
11
12 int n;
13 int m;
14
15
16 GLuint *image;
17
18
19 void display()
20 {
21 glClear(GL_COLOR_BUFFER_BIT);
22
23 glBegin(GL_QUADS);
24 glTexCoord2f(0.0,0.0);
25 glTexCoord2f(s, t);
26
27 glTexCoord2f(s, t);
28
29 glTexCoord2f(s, t);
30
31 glVertex2i(n-1,0);
32 glEnd();
33 glFlush();
34 }
35
36
37 void myreshape(int h, int w)
38 {
39 glMatrixMode(GL_PROJECTION);
ut from:

Interpolation

OpenGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

good selection of **tex coordinates**

poor selection of **tex coordinates**

texture stretched over trapezoid showing effects of **bilinear interpolation**

Texture Parameters

OpenGL has a variety of parameters that determine how exture is applied

Wrapping parameters determine what happens if **s** and **t** are outside the (0,1) range

Filter modes allow us to **use area averaging** instead of **point samples**

Mipmapping allows us to **use textures at multiple resolutions**

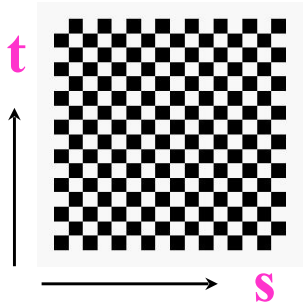
Environment parameters determine **how texture mapping interacts with shading**

Wrapping Mode

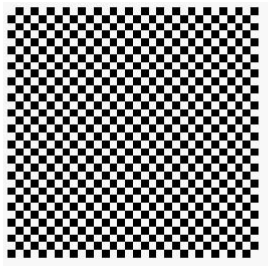
Clamping: if s,t > 1 use 1, if s,t <0 use 0

Wrapping: use s,t modulo 1

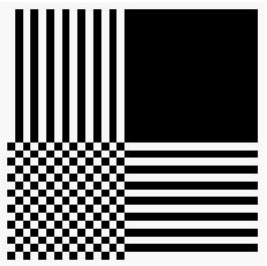
```
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP)
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT)
```



texture



GL_REPEAT
wrapping

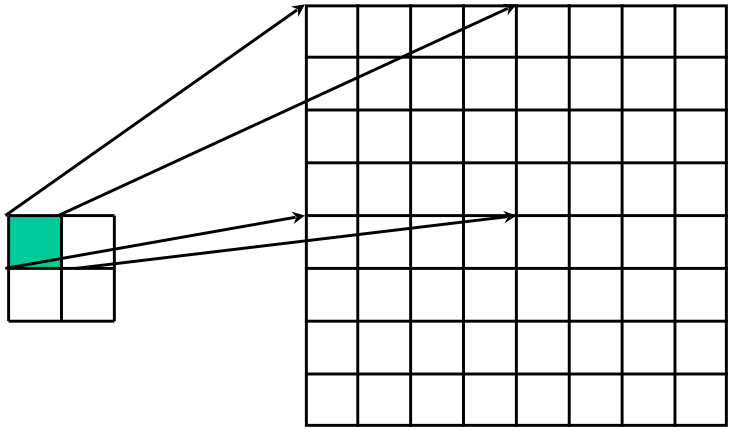


GL_CLAMP
wrapping

Magnification and Minification

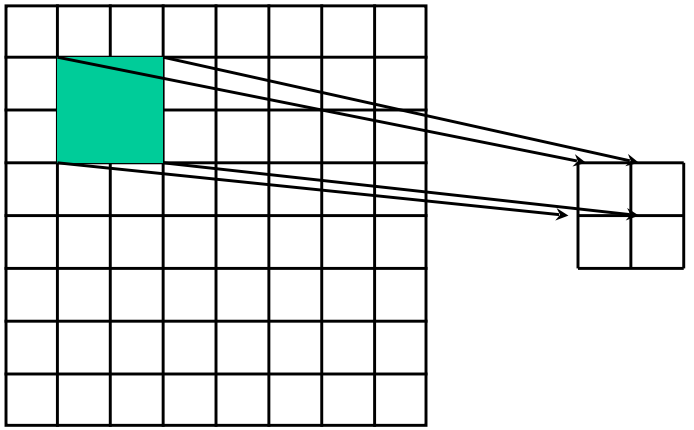
More than one pixel can cover a texel (***magnification***) or
More than one texel can cover a pixel (***minification***)

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



Texture Polygon

Magnification



Texture Polygon

Minification

Filter Modes

Modes determined by

```
glTexParameteri(target, type, mode )
```

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXURE_MAG_FILTER,GL_NEAREST);
```

```
glTexParameteri(GL_TEXTURE_2D,GL_TEXURE_MIN_FILTER,GL_LINEAR);
```

Note that linear filtering requires a border of an **extra texel for filtering at edges (border = 1)**

Debug | Win32 | texcoord

ppmtex.c

(Global Scope) | main(int argc, char ** argv)

```
94      printf("read image\n");
95      glutInit(&argc, argv);
96      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
97      glutInitWindowSize(n, m);
98      glutInitWindowPosition(0,0);
99      glutCreateWindow("ppm textured image");
100     glutReshapeFunc(myreshape);
101     glutDisplayFunc(display);
102     glPixelTransferf(GL_RED_SCALE, s);
103     glPixelTransferf(GL_GREEN_SCALE, s);
104     glPixelTransferf(GL_BLUE_SCALE, s);
105     glPixelStorei(GL_UNPACK_SWAP_BYTES,GL_TRUE);
106     glPixelStorei(GL_UNPACK_ALIGNMENT,1);
107     glEnable(GL_TEXTURE_2D);
108     glTexImage2D(GL_TEXTURE_2D,0,3,n,m,0,GL_RGB,GL_UNSIGNED_INT, image);
109     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
110     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
111     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
112     glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D,GL_TEXURE_MAG_FILTER,GL_NEAREST)
        glTexParameteri(GL_TEXTURE_2D,GL_TEXURE_MIN_FILTER,GL_LINEAR);
116
117
118 }
119
120
121
```

Mipmapped Textures

Mipmapping allows for prefiltered texture maps of decreasing resolutions

Lessens interpolation errors for smaller textured objects

Declare mipmap level during **texture definition**

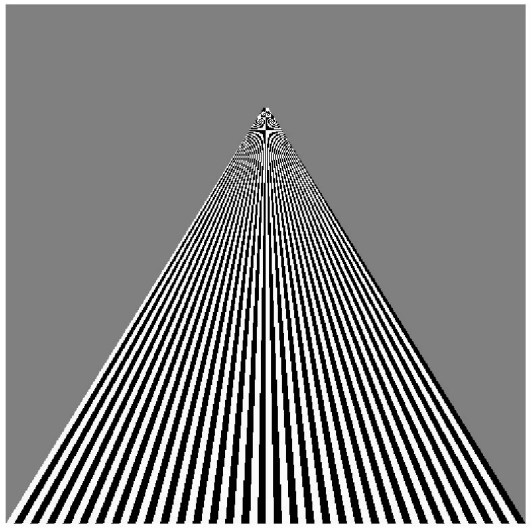
```
glTexImage2D( GL_TEXTURE_*D, level, … )
```

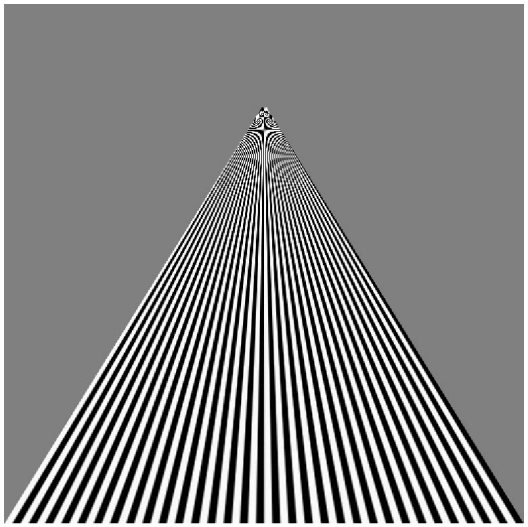
GLU mipmap builder routines will **build all the textures from a given image**

```
gluBuild*DMipmaps( … )
```

Example
Object is a quadrilateral that looks almost as a triangle

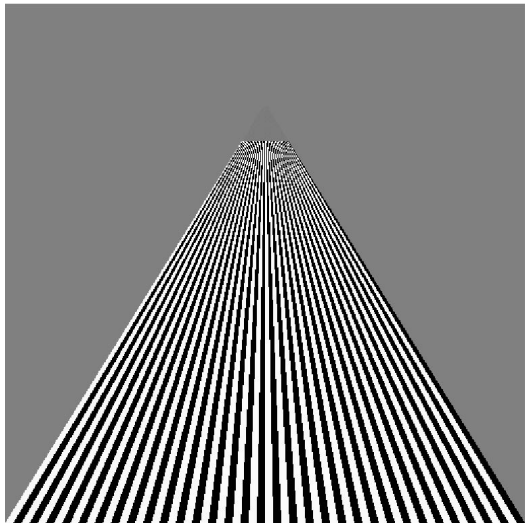
point sampling

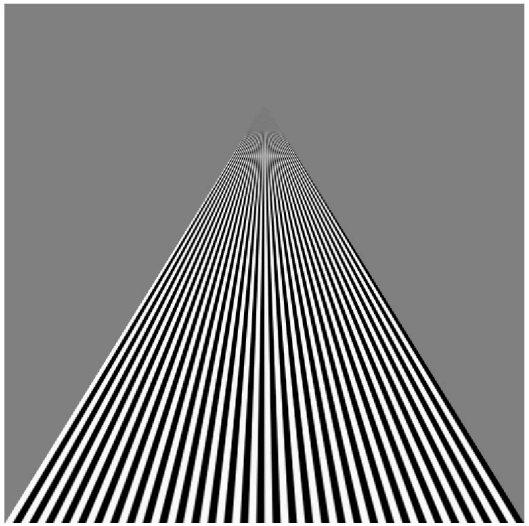




linear filtering

mipmapped point sampling





mipmapped linear filtering

The texture is a series of black and white lines that appear to converge

Texture & Shading Functions

Controls how texture is applied

```
glTexEnv{fi}[v]( GL_TEXTURE_ENV,prop,param)
```

GL_TEXTURE_ENV_MODE modes

- **GL_MODULATE**: modulates with computed shade
- **GL_BLEND**: blends with an environmental color
- **GL_REPLACE**: use only **texture** color

```
GL(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
GL_MODULATE);
```

Set blend color with **GL_TEXTURE_ENV_COLOR**

Perspective Correction Hint

Texture coordinate and color interpolation

Proper **texture mapping** also depends on what **type of projection** is used. Normally, **OpenGL** uses bilinear interpolation in **screen space** to find a **texture value**. **For orthogonal projections**, the bilinear map is correct, but it is not correct for **perspective projections** because of the nonlinear depth scaling.

Noticeable for polygons "on edge"

```
glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )
```

where **hint** is one of

- GL_DONT_CARE
- GL_NICEST - default
- GL_FASTEST

Texture Coordinates

- Assumed rectangular polygons then we can generate texture coordinates
- Textures coordinates form a 4X4 texture matrix. Can be manipulated in the same manner as the model-view and projection matrices.
- We use it by setting the current matrix mode as follows:

```
glMatrixMode(GL_TEXTURE);
```

- We can use this matrix to scale and orient textures and to create effects in which the texture moves with the object, the camera, or the lights.

Generating Texture Coordinates
curved objects

OpenGL can generate texture coordinates automatically

`glTexGen{ifd}[v]()`

specify a plane

generate **texture coordinates based upon distance from the** plane

generation modes

`GL_OBJECT_LINEAR`

`GL_EYE_LINEAR`

`GL_SPHERE_MAP` (used for **environmental maps**)

Texture Objects

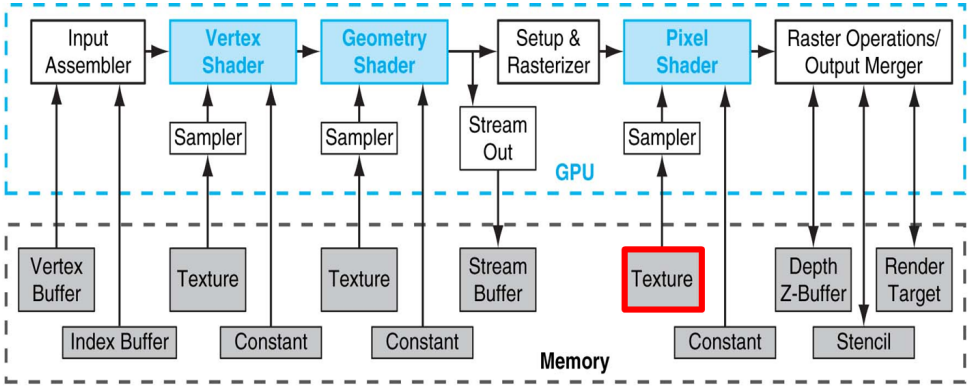
- Texture is part of the **OpenGL state**

  If we have different **textures** for different **objects**, **OpenGL** will be moving large amounts data from **processor memory** to **texture memory**

- Recent versions of **OpenGL** have ***texture objects***

  one image per **texture object**

  Texture memory can hold **multiple texture objects**

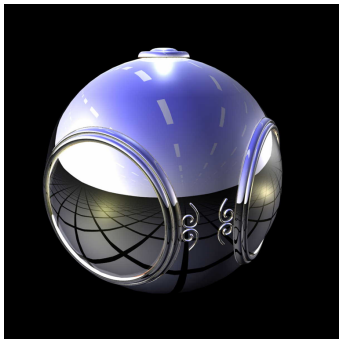


Applying Textures II

1. specify **textures** in **texture objects**
2. set texture filter
3. set texture function
4. set texture wrap mode
5. set optional perspective correction hint
6. bind **texture object**
7. enable texturing
8. supply texture coordinates for **vertex**

   coordinates can also be generated

Other Texture Features



Environment Maps (Highly reflective surfaces are characterized by **specular reflections** that mirror the environment**)**

Start with **image of environment** through a wide angle lens

- Can be either a **real scanned image** or an **image** created in **OpenGL**

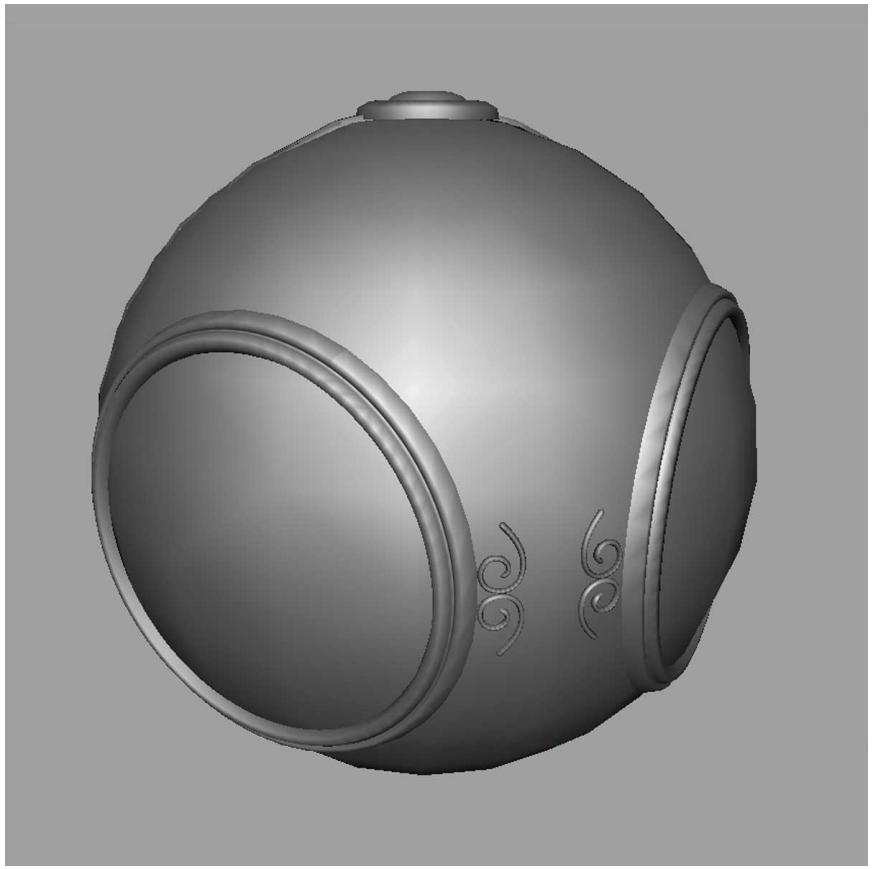
Use **this texture** to generate a **spherical map**

Use **automatic texture coordinate generation**

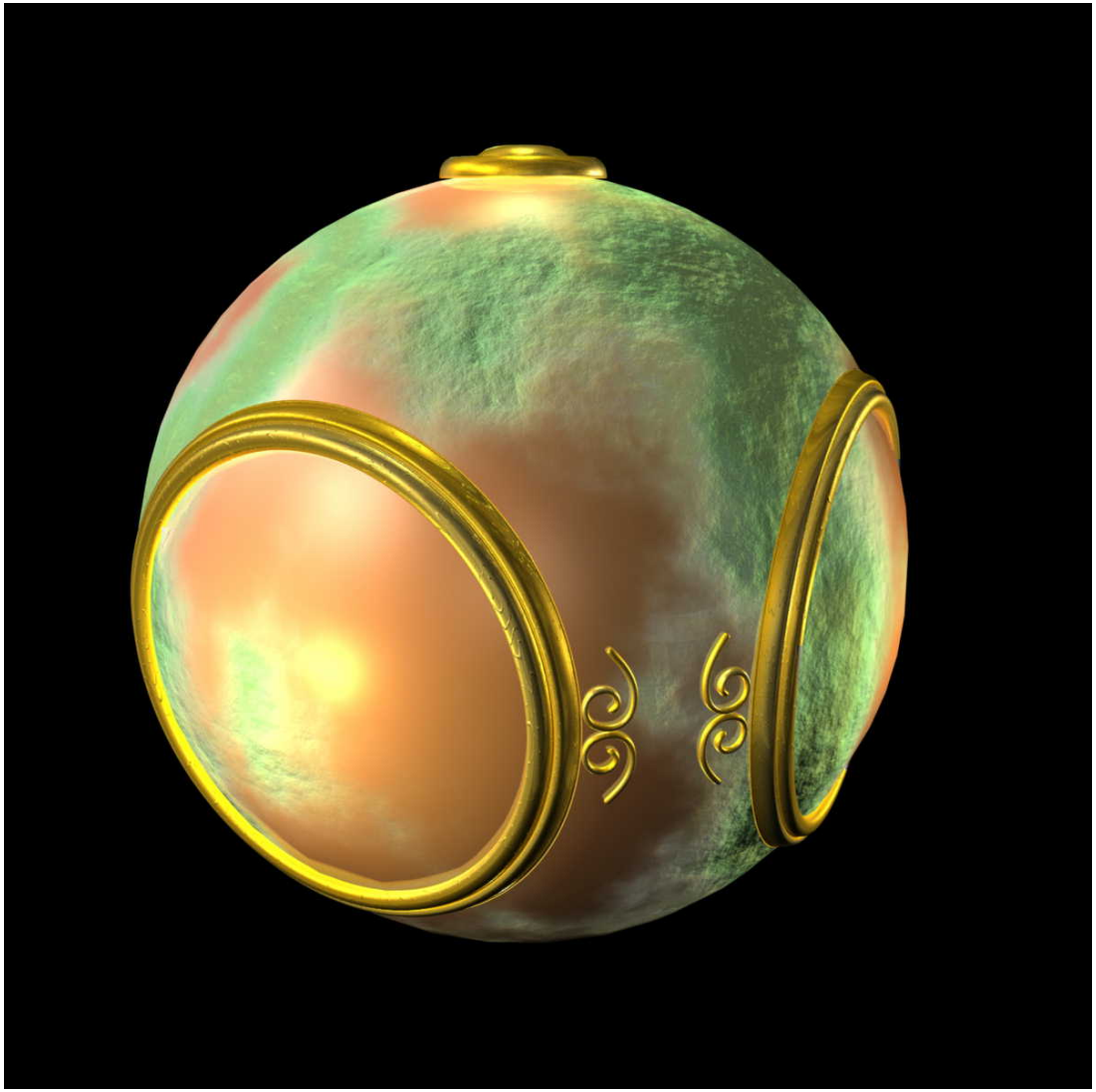
Multitexturing

Apply a **sequence of textures** through cascaded **texture** units

Bump Mapping



geometric model



Compositing and Blending

Chapter 8.11

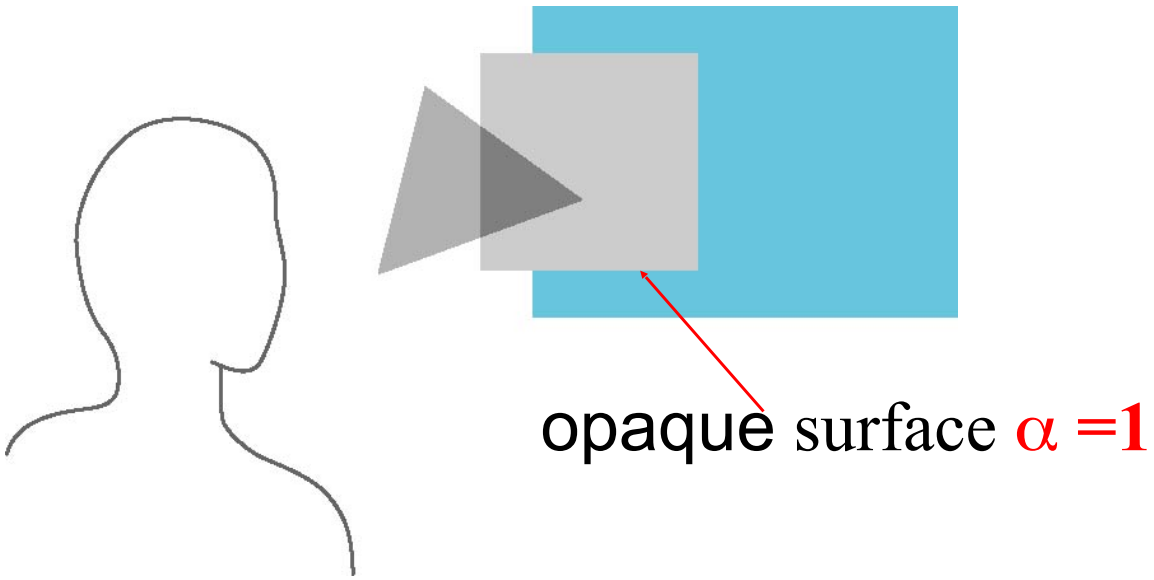
Objectives

- Learn to use the A component in RGBA color for
  - **Blending for translucent surfaces**
  - **Compositing images**
  - **Antialiasing**

Opacity and Transparency

- **Opaque surfaces** permit no light to pass through
- **Transparent surfaces** permit all light to pass
- **Translucent surfaces** pass some light

translucency = 1 – opacity (α)

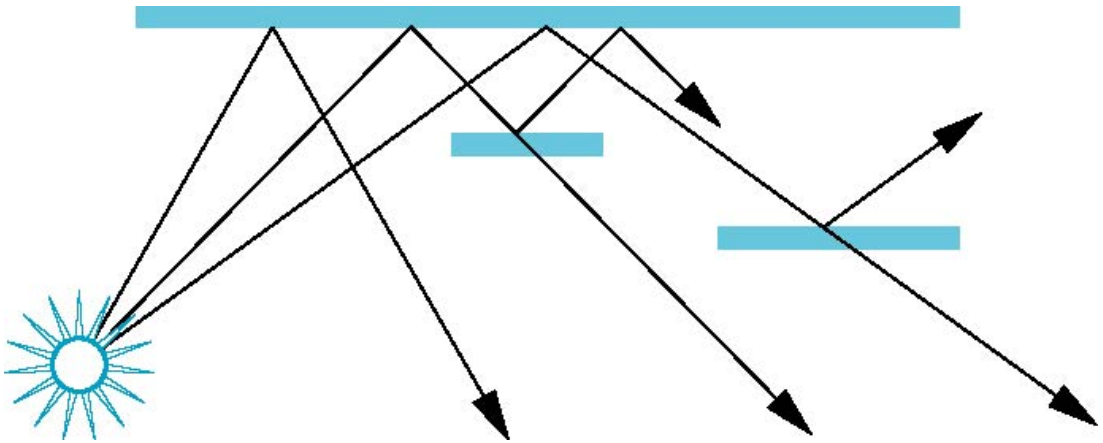


Physical Models

Dealing with translucency in a physically correct manner is difficult due to

the **complexity of the internal interactions of light and matter**

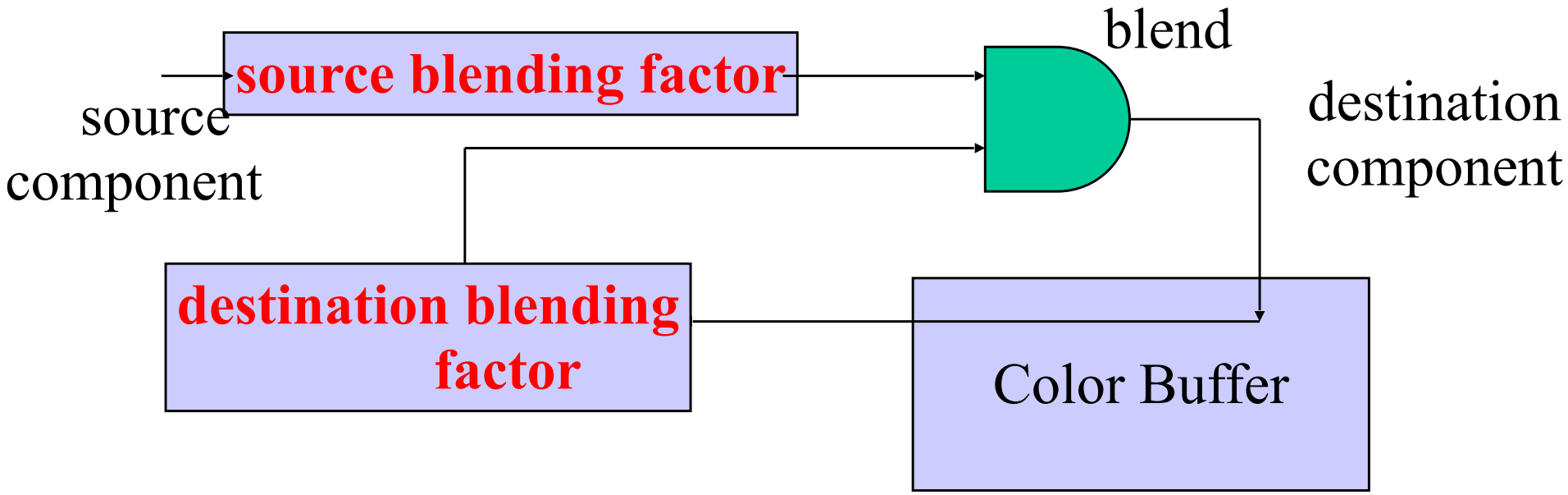
Using a **FORWARD pipeline renderer**



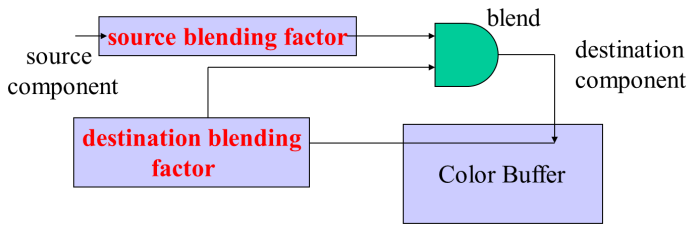
Writing Model

Use **A** component of RGB**A** (or RGBα) color to store opacity

During rendering we can expand our **writing model to use RGBA values**



Blending Equation



- We can define source and destination blending factors for each RGBA component

$$\mathbf{s} = [s_r, s_g, s_b, s_\alpha]$$

$$\mathbf{d} = [d_r, d_g, d_b, d_\alpha]$$

Suppose that the source and destination colors are

$$\mathbf{b} = [b_r, b_g, b_b, b_\alpha]$$

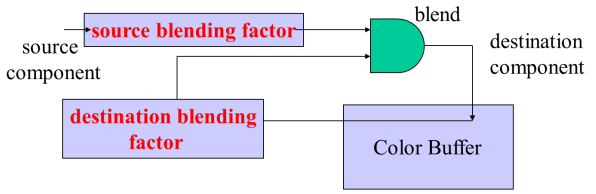
$$\mathbf{c} = [c_r, c_g, c_b, c_\alpha]$$

Blend as

$$\mathbf{c}' = [b_r s_r + c_r d_r, b_g s_g + c_g d_g, b_b s_b + c_b d_b, b_\alpha s_\alpha + c_\alpha d_\alpha]$$

OpenGL Blending and Compositing

- Must enable blending and pick source and destination factors




```
glEnable(GL_BLEND)
glBlendFunc(source_factor,destination_factor)
```

- Only certain factors supported

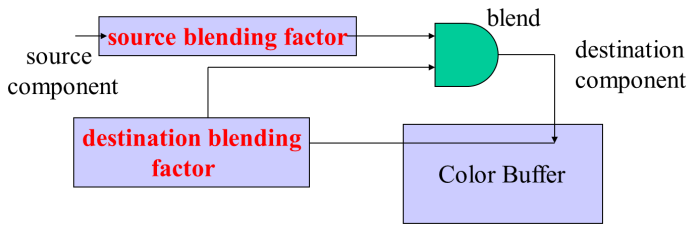
  GL_ZERO, GL_ONE

  GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA

  GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA

  See Redbook for complete list

Example



146

- Suppose that we start with the **opaque background color** $(R_0,G_0,B_0,1)$

  This color becomes the initial destination color

- **We now want to blend in a translucent polygon with color (R_1,G_1,B_1,α_1)**
- Select `GL_SRC_ALPHA` and `GL_ONE_MINUS_SRC_ALPHA` as the source and destination blending factors

  $R'_1 = \alpha_1 R_1 + (1- \alpha_1) R_0$, ……

- Note this formula is correct if polygon is either **opaque** or **transparent**

Clamping and Accuracy

All the components (RGB**A**) are **clamped** and stay in the range (0,1)

However, in a typical system, RGB**A** values are only stored to 8 bits

Can easily **loose accuracy** if we add many components together

Example: add together **n images**

- Divide all color components by **n** to avoid **clamping**
- **Blend** with source factor = 1, destination factor = 1
- **But division by n loses bits**

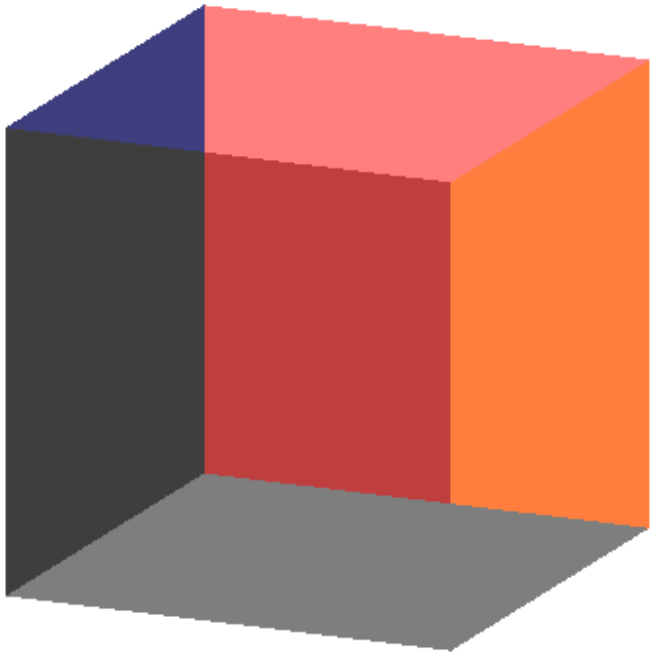
Order Dependency

- **Is this image correct?**

  Probably not

  Polygons are rendered **in the order they pass down the pipeline**

  Blending functions are order **dependent**



Opaque and Translucent Polygons

- Suppose that we have a **group of polygons** some of which are opaque and some translucent

- **How do we use hidden-surface removal?**
- **Opaque polygons** block all polygons behind them and affect the **depth buffer**

- **Translucent polygons** should not affect **depth buffer**

  Render with `glDepthMask(GL_FALSE)` which makes **depth buffer** read-only

- **Sort polygons first to remove order dependency**

Fog

We can **composite** with a fixed color and have the **blending factors depend on depth**

Simulates a fog effect

Blend source color C_s and fog color C_f by

$$C_s' = f\, C_s + (1-f)\, C_f$$

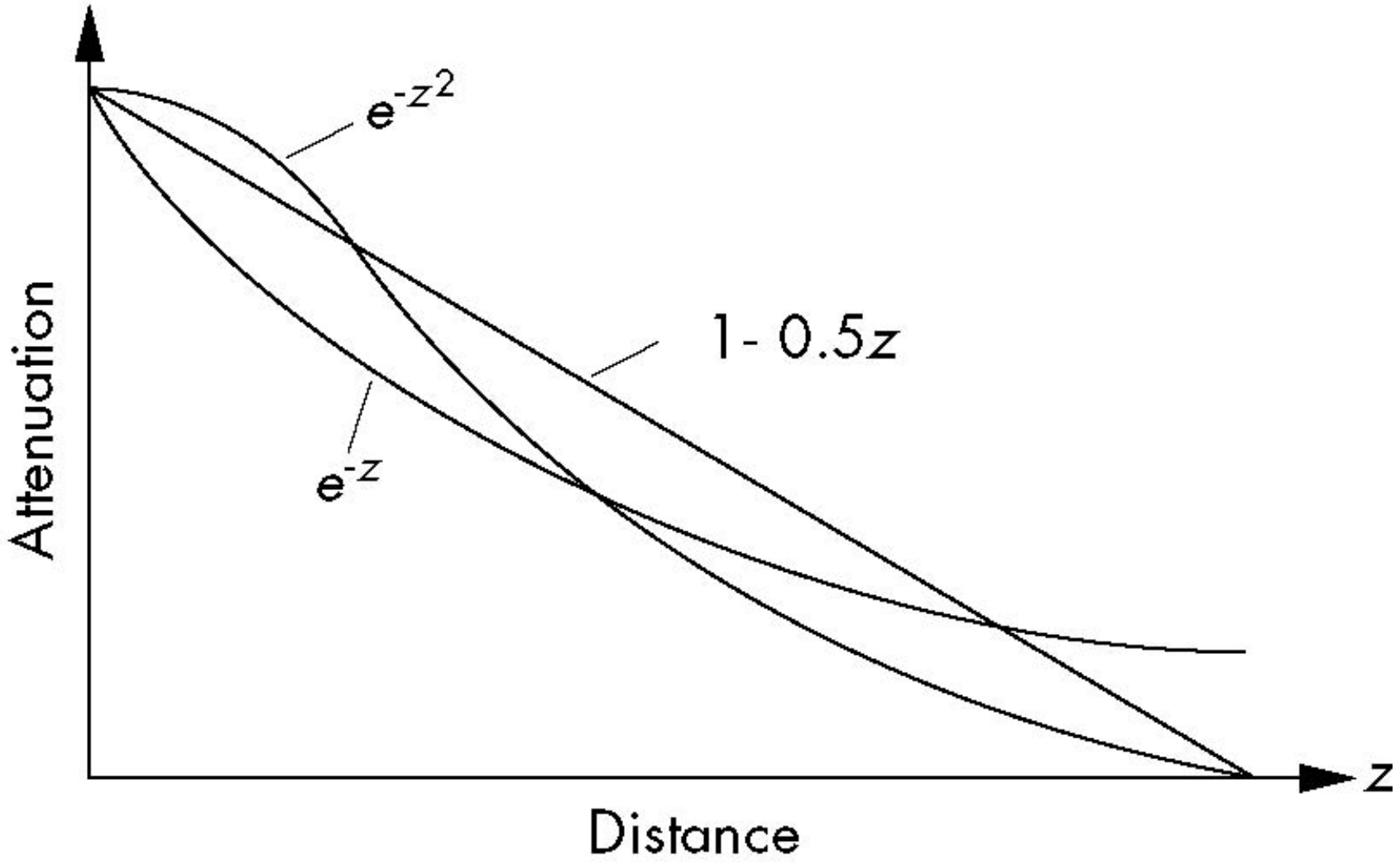
f is the *fog factor*

Exponential

Gaussian

Linear (depth cueing)

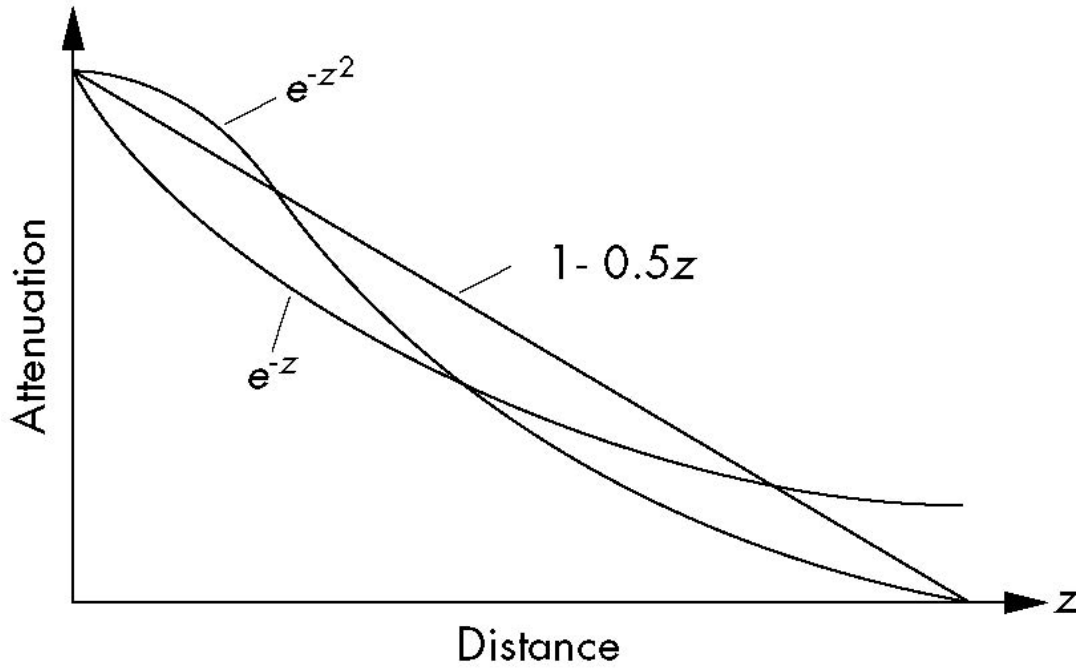
Fog Functions



OpenGL Fog Functions

```
GLfloat fcolor[4] = {……}:

glEnable(GL_FOG);
glFogf(GL_FOG_MODE, GL_EXP);
glFogf(GL_FOG_DENSITY, 0.5);
glFOgv(GL_FOG, fcolor);
```



Line and Polygon Aliasing

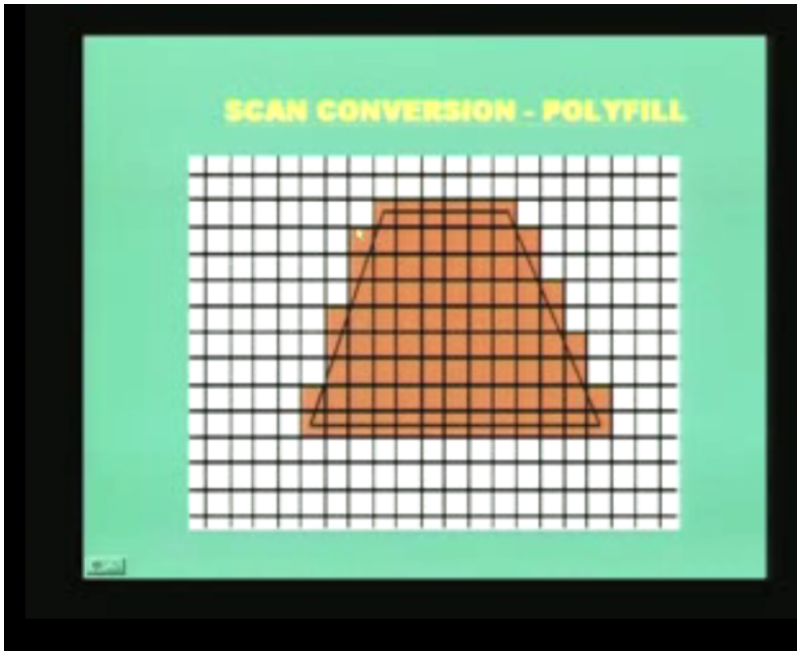
Ideal raster line is one pixel wide

All line segments, other than vertical and horizonta segments, **partially cover pixels**

Simple algorithms color only whole pixels

Lead to the “jaggies” or aliasing

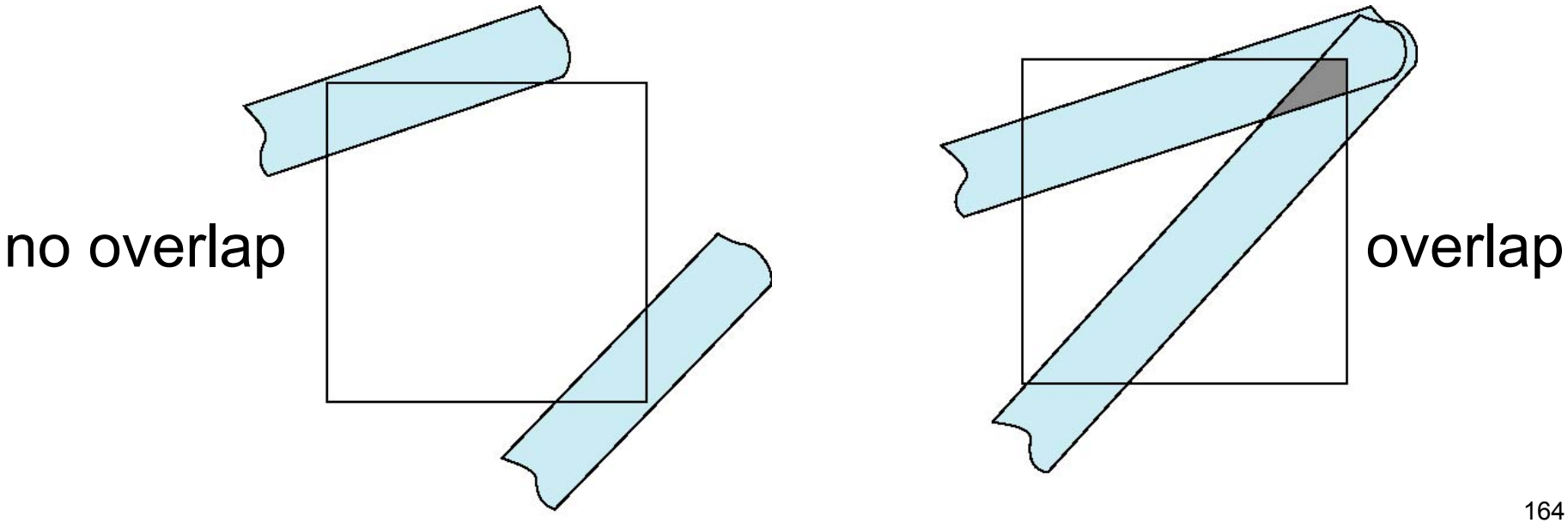
Similar issue for polygons



Antialiasing

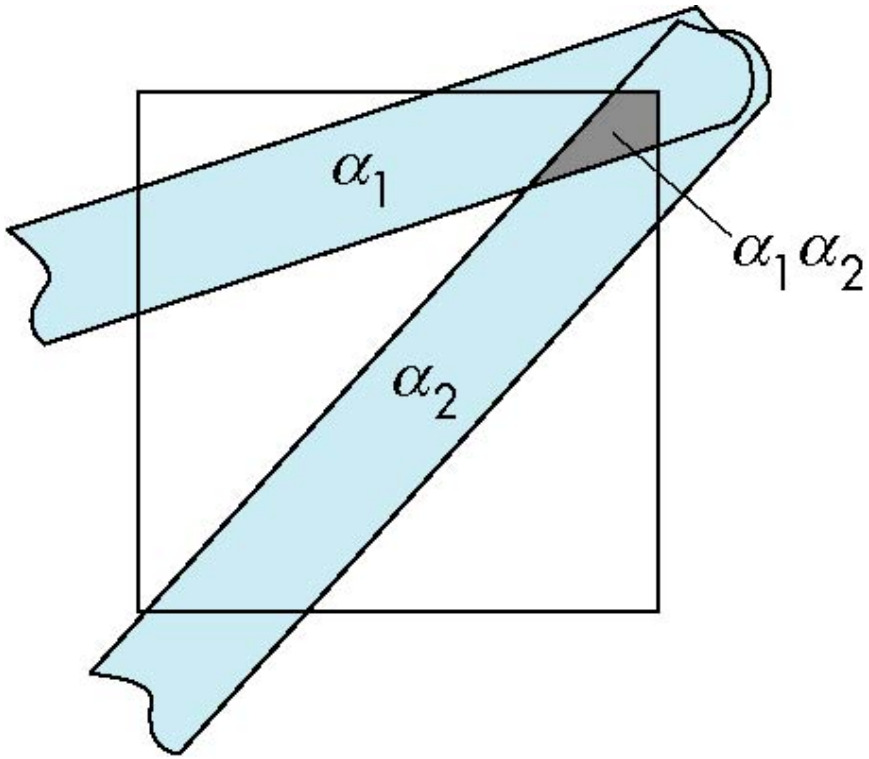
Can try to color a pixel by adding a fraction of its color to the frame buffer

- **Fraction depends on percentage of pixel covered by fragment**
- **Fraction depends on whether there is overlap**



Area Averaging

- **Use average area $\alpha_1+\alpha_2-\alpha_1\alpha_2$ as blending factor**



OpenGL Antialiasing

Can enable separately for points, lines, or polygons

```
glEnable(GL_POINT_SMOOTH);
glEnable(GL_LINE_SMOOTH);
glEnable(GL_POLYGON_SMOOTH);

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

NEXT.

10.23.2023 (M 5:30 to 7) (18)		**Lecture 9** **(Programmable Shaders)**
10.25.2023 (W 5:30 to 7) (19)		**Lecture 10** **(Modeling and Hierarchy)**
10.30.2023 (M 5:30 to 7) (20)		**PROJECT 3**
11.01.2023 (W 5:30 to 7) (21)		**EXAM 3 REVIEW**
11.06.2023 (M 5:30 to 7) (22)		**EXAM 3**

At 6:45 PM.

End Class 17

VH, Download Attendance Report
Rename it:
10.18.2023 Attendance Report FINAL

VH, upload Lecture 8 to CANVAS.