# Texture and Other Mapping Techniques

# Intended Learning Outcomes

- Able to apply pixel order scanning for generating texture
- Describe and apply other advanced mapping methods

# Two methods of texture mapping

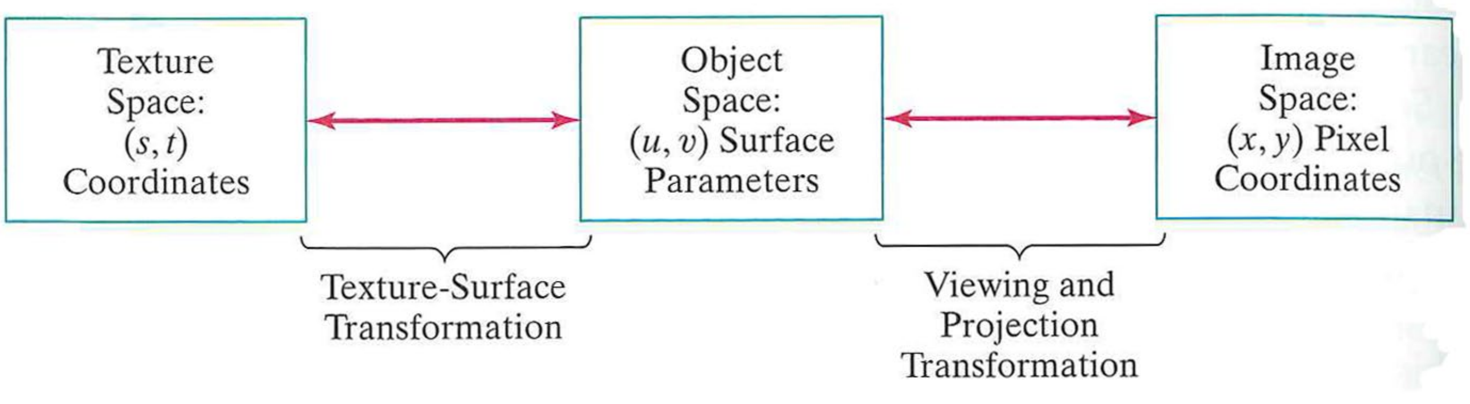
- <u>Texture scanning</u> : map texture pattern in (s, t) to pixel (x, y). Left to right in Fig. below

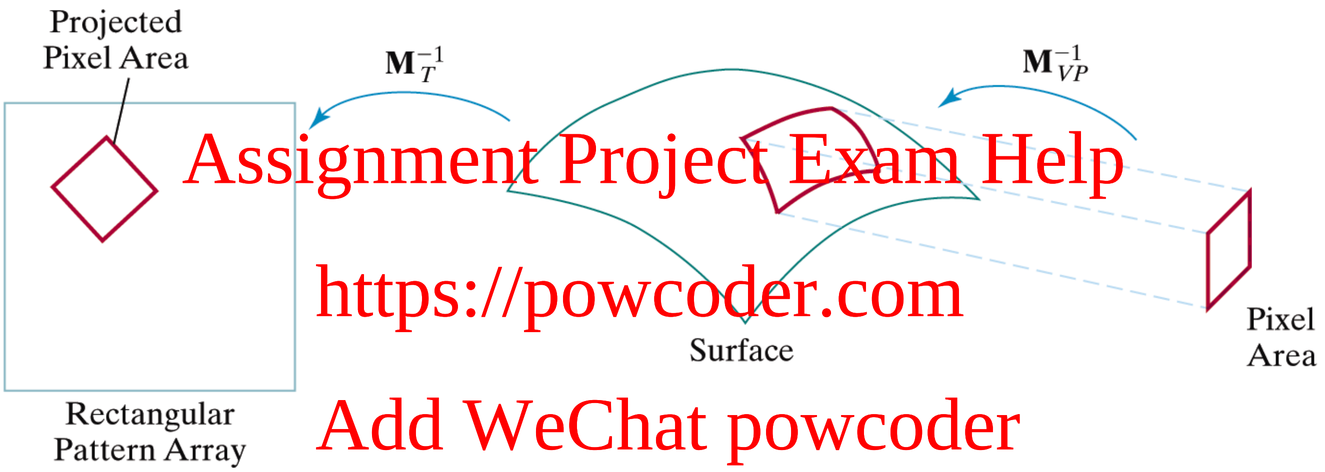


- <u>pixel order scanning</u> : map pixel (x,y) to texture pattern in (s, t). Right to left in Fig. below












Pixel order scanning

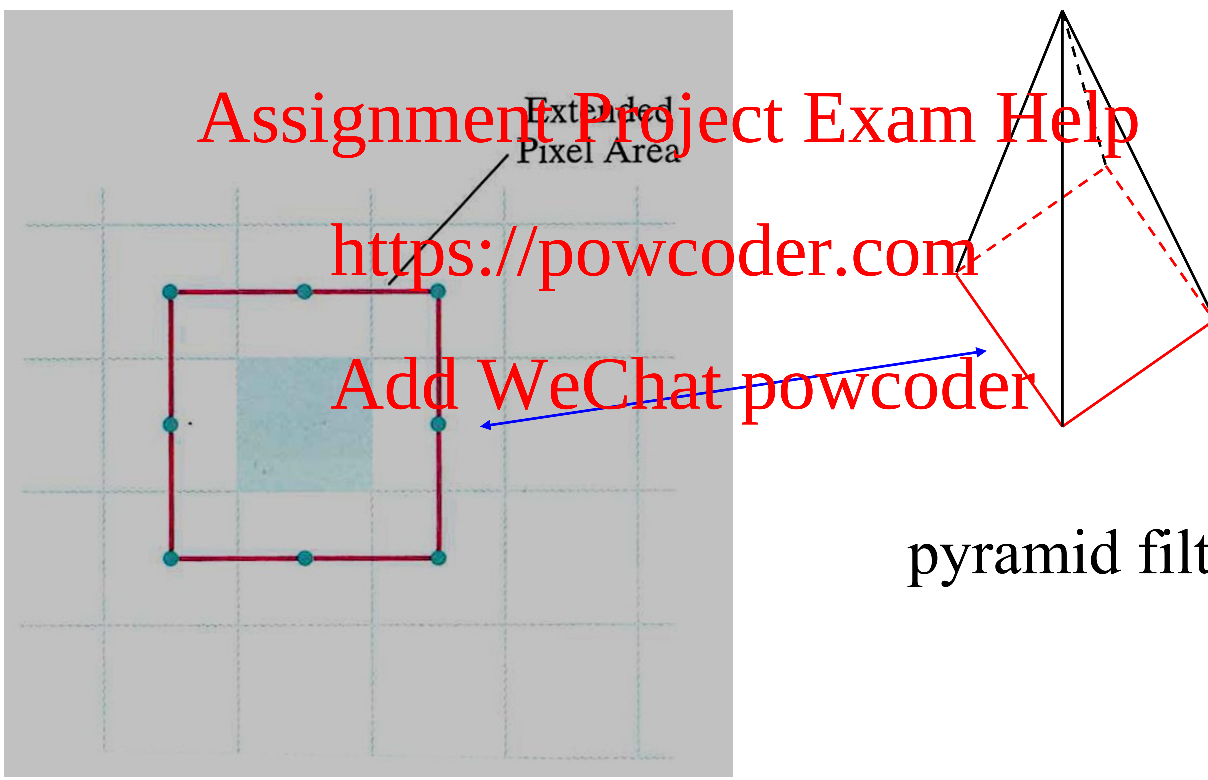
- To simplify calculations, the mapping from texture space to object space is often specified with linear functions:

$$u = f_u(s,t) = a_u s + b_u t + c_u$$
$$v = f_v(s,t) = a_v s + b_v t + c_v$$



- The mapping from object space to image space consists of a concatenation of 1) viewing transformation followed by 2) projective transformation.

- Texture mapping is not used in practice. Pixel order scanning is used, together with antialiasing, as shown below:




pyramid filter

# Example: Pixel Order Scanning

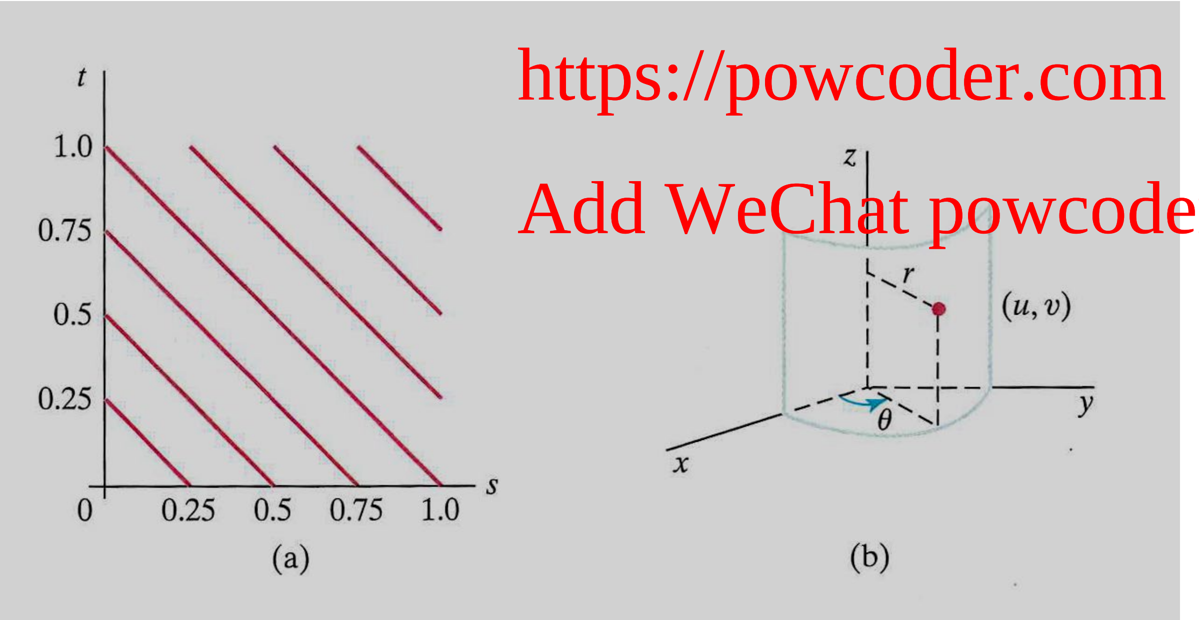
- Map texture pattern in Fig. (a) to the cylindrical surface in Fig. (b).
- Parametric representation of the cylindrical surface:



$$X = r\cos u$$

$$Y = r\sin u$$

$$Z = v$$



- Map the texture pattern to the surface by defining the following linear function

$$u = \frac{\pi}{2} s \qquad (1)$$

$$v = t$$



- The above is the texture-surface transformation $M_T$
- Suppose no geometrical transformation and projection is orthographic with projection direction in the X direction. Then Y-Z is the projection plane
- Viewing and projection transformation $M_{VP}$ is

$$Y = r \sin u \qquad (2)$$

$$Z = v$$

- For pixel order scanning, we need to compute the transformation (Y,Z)→(s, t)
- First compute $\mathbf{M}_{VP}^{-1}$, or (Y, Z)→(u, v). From (2)

$$u = \sin^{-1}\left(\frac{Y}{r}\right)$$
$$v = Z \qquad (3)$$



- Next compute $\mathbf{M}_{T}^{-1}$, or (u, v)→(s, t). From (1)



$$s = \frac{2}{\pi}u$$
$$t = v \qquad (4)$$

- Combining (3) and (4)

$$s = \frac{2}{\pi} \sin^{-1}\left(\frac{Y}{r}\right)$$

$$t = Z$$

- Using this transformation, the pixel area of a pixel (Y, Z) will be back-transformed into an area in the texture space (s, t). Intensity values in this area are averaged to obtain the pixel intensity.

# Bump Mapping

- Texture mapping can be used to add fine surface detail to smooth surface. However, it is not a good method for modelling rough surface e.g., oranges, strawberries, since the illumination detail in the texture pattern usually does not correspond to the illumination direction in the scene.
- Bump mapping is a method for creating surface bumpiness. A perturbation function is applied to the surface normal. The perturbed normal is used in the illumination model calculations.

**P**(u, v) position on a parametric surface

**N** surface normal at (u, v)

$\mathbf{N} = \mathbf{P}_u \times \mathbf{P}_v$

where $$P_u = \frac{\partial P}{\partial u} \qquad P_v = \frac{\partial P}{\partial v}$$



Add a small bump function b(u, v) to **P**(u, v). It becomes





**P**(u,v) + b(u,v)**n**

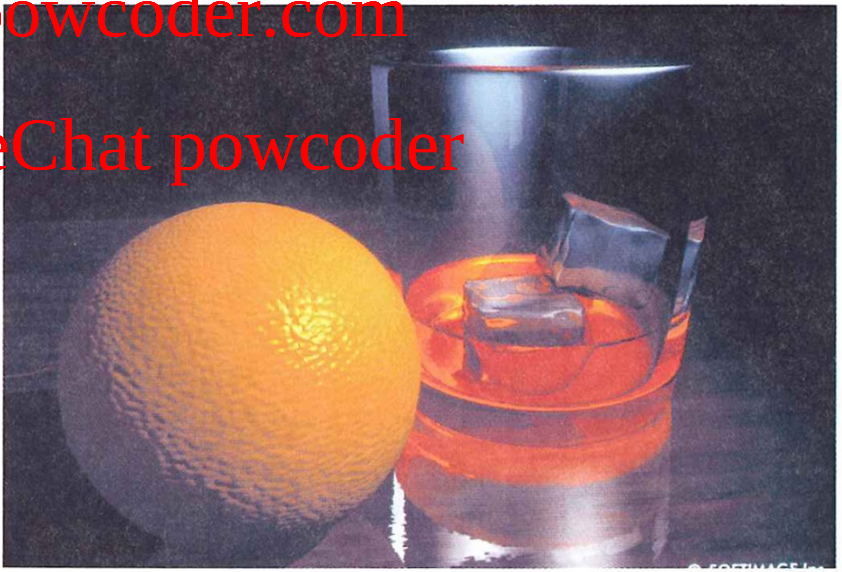
where **n** = **N** / |**N**| is the unit (outward) surface normal

The normal $\mathbf{N} = \mathbf{P}_u \times \mathbf{P}_v$ is perturbed.

- The bump function b(u, v) are usually obtained by table lookup. It can be setup using

1) Random pattern to model irregular surfaces (e.g. raisin)
2) Repeating pattern to model regular surfaces (e.g. orange Fig. 10-110)

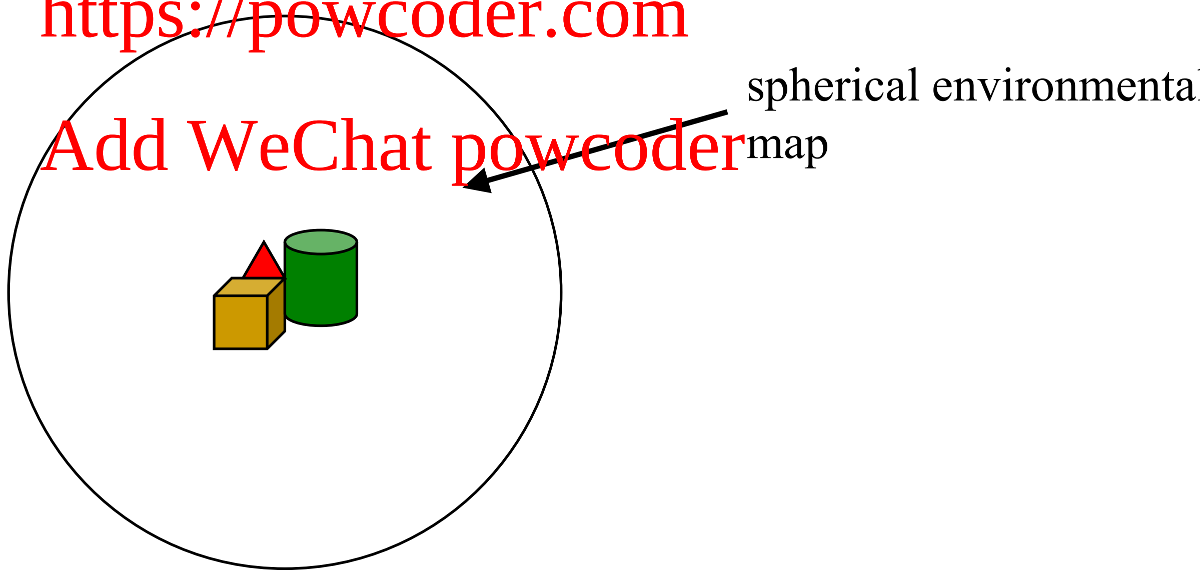




# Environment Mapping

- A simplified ray tracing method that uses texture mapping concept.
- Environment map is defined over the surface of an enclosing universe. Information includes intensity values of light sources, the sky or other background objects.

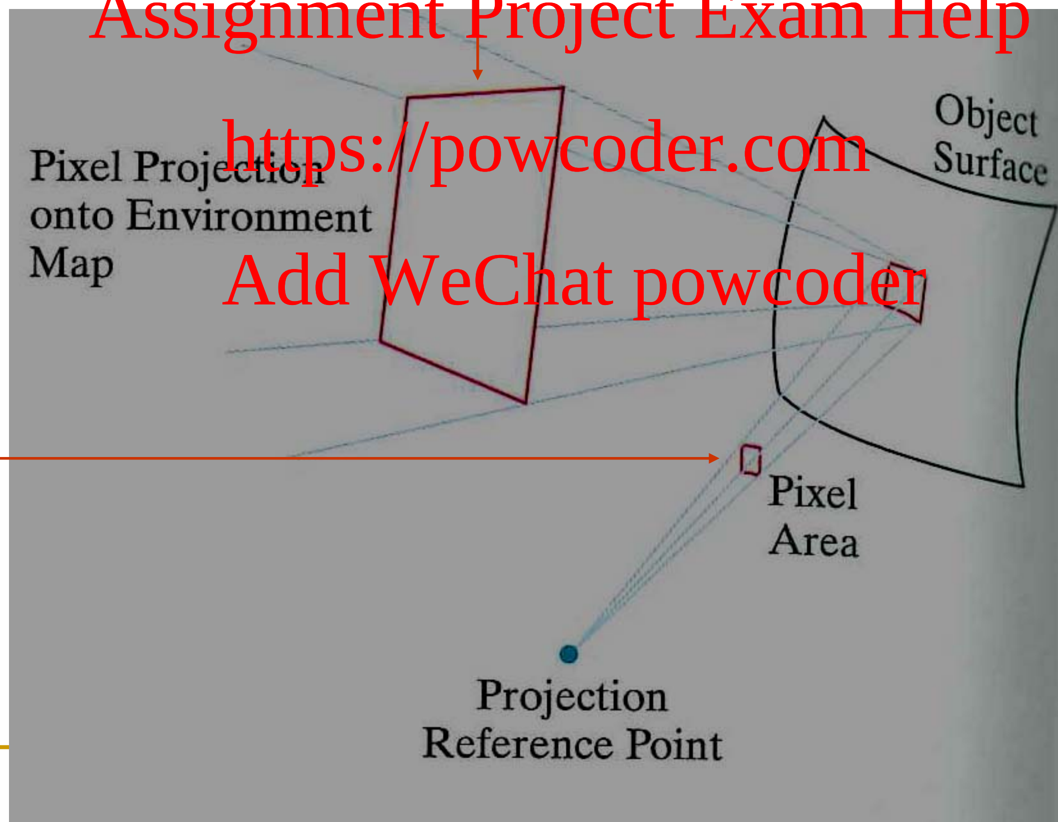


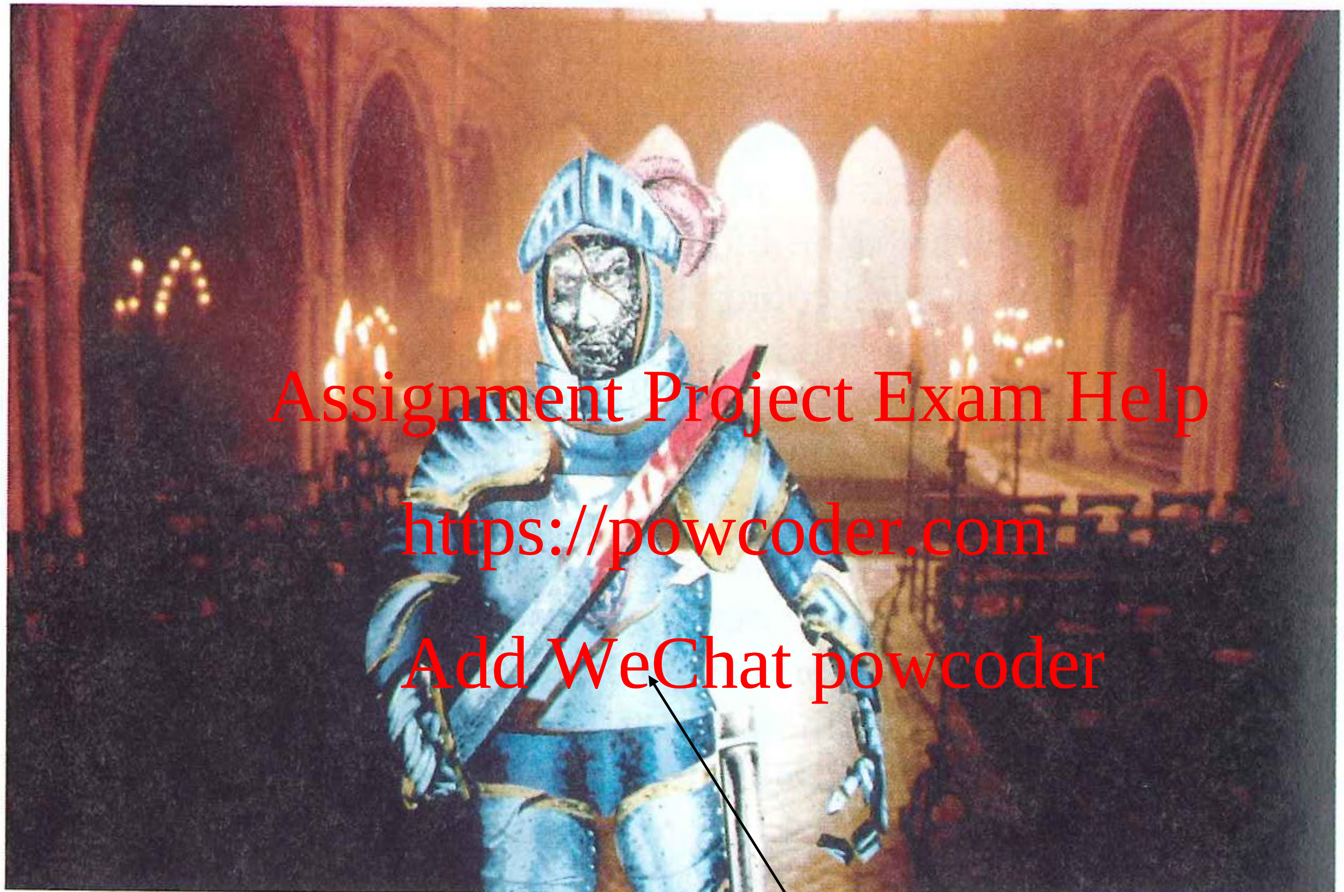



- Run “Example environment map”

- A surface is rendered by projecting the pixel area to the surface, then reflect onto the environment map. If the surface is transparent, also refract onto the map.
- Pixel intensity determined by averaging the intensity values within the intersected region of the environment map.









armour (specular object) reflects the cathedral surrounding
Modelled using environmental map

# OpenGL functions

*glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, texWidth, texHeight, 0, dataFormat, dataType, surfTexArray);*



*GL_RGBA* Each colour of the texture pattern is specified with *(R, G, B, A)* *A* is the alpha parameter:



$A = 1.0 \Rightarrow$ completely transparent

$A = 0.0 \Rightarrow$ opaque



*texWidth* and *texHeight* is the width and height of the pattern

*dataFormat* and *dataType* specify the format and type of the texture pattern e.g. *GL_RGBA* and *GL_UNSIGNED_BYTE* *glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)*

*glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)*

Specify what to do if the texture is to be magnified (i.e., mag) or reduced (i.e., min) in size:

| | |
|---|---|
| *GL_NEAREST* | assigns the nearest texture colour |
| *GL_LINEAR* | linear interpolate |

*glTexCoord2* ( sCoord, tCoord );*

A texture pattern is normalized such that s and t are in |0, 1|

A coordinate position in 2-D texture space is selected with

$0.0 \leq sCoord, tCoord \leq 1.0$



*glEnable (GL_TEXTURE_2D)*

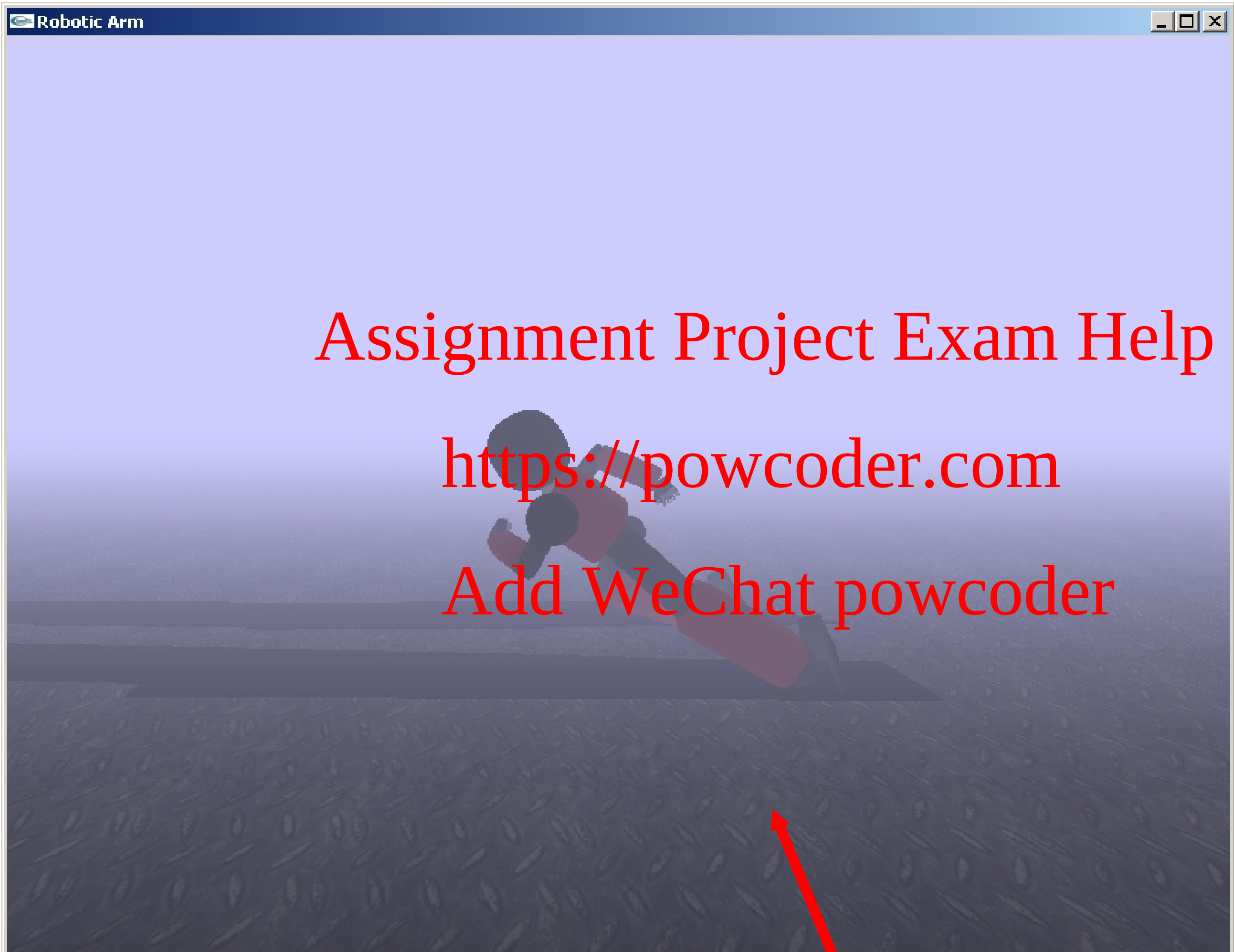
*glDisable (GL_TEXTURE_2D)*

Enables / disables texture

# Example: texture map a quadrilateral

```
GLubyte texArray [808][627][4];

glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, 808, 627, 0, GL_RGBA,
GL_UNSIGNED_BYTE, texArray);

glEnable (GL_TEXTURE_2D);

// assign the full range of texture colors to a quadrilateral
glBegin (GL_QUADS);
        glTexCoord2f (0.0, 0.0);   glVertex3fv (vertex1);
        glTexCoord2f (1.0, 0.0);   glVertex3fv (vertex2);
        glTexCoord2f (1.0, 1.0);   glVertex3fv (vertex3);
        glTexCoord2f (0.0, 1.0);   glVertex3fv (vertex4);
glEnd ( );

glDisable (GL_TEXTURE_2D);
```

# Simple example






Use a large QUAD for the ground and texture map it

- To re-use the texture, we can assign a name to it

```
static GLuint    texName;
glGenTextures (1, &texName);  // generate 1 texture with name "texName"

glBindTexture (GL_TEXTURE_2D, texName);
glTexImage2D (GL_TEXTURE_2D, 0, GL_RGBA, 32, 32, 0, GL_RGBA,
GL_UNSIGNED_BYTE, texArray);    // define the texture "texName"

                        ⋮

glBindTexture (GL_TEXTURE_2D, texName);  // use it as current texture
```

- We can generate more than 1 name at a time. To generate 6 names:

*static Gluint texNamesArray [6];*

*glGenTexures (6, texNamesArray);  // generate 6 texture names*

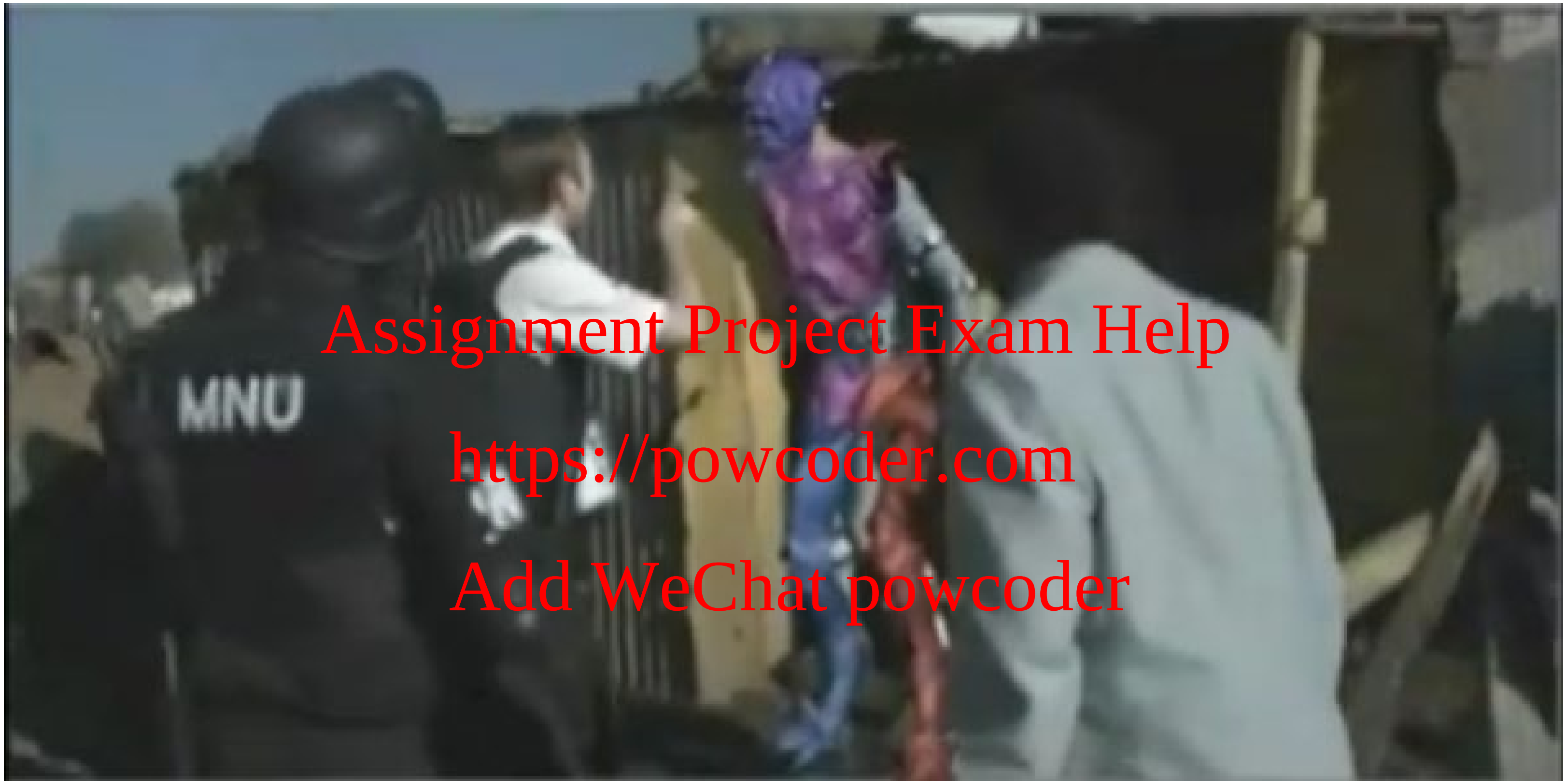


- To use *texNamesArray [3]*



*glBindTexture  (GL_TEXTURE_2D, texNamesArray [3]);*

# Texture mapping in Movie



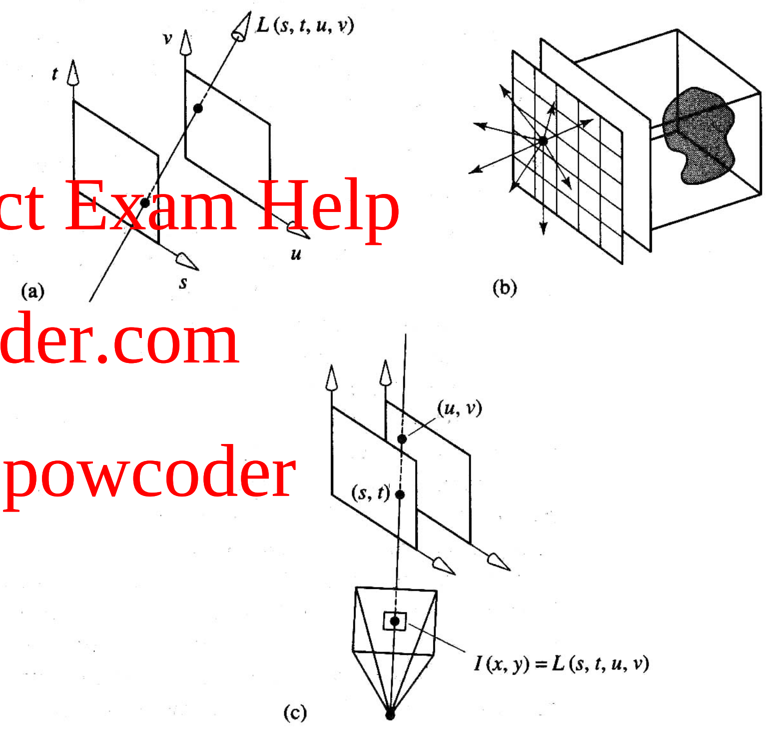



- Use texture map to blend graphics object into real movie production
- Double buffering is used
- Frame rate is unimportant as movie is produced off-line
- Human artist can optionally  help with later stage production to make image more realistic

# Light field (Lumigraph)

- An image based rendering (IBR) approach
- A “pre-computation” idea
- Stores intensity of all rays in all directions
- Uses data compression

- Adv.: Extremely fast
- Disadv.: High Pre-computational cost



# Application

- Light field camera

  https://en.wikipedia.org/wiki/Light-field_camera



- Capture instantly. Do not need to focus.

# References

- Text  Ch. 18 on Texture
- Text  Ch. 21-3 on Environment Mapping
- Light field: A. Watt, 3D Computer Graphics, 3rd Ed. (2000) pp. 463-65

# Implementation notes

- One may use OpenGL SOIL library or stb_image.h for reading in texture images
- Search the web with keyword “texture images”
- A .raw file is a file with no formatting and only consist of a sequence of numbers. You can read the file into an array in C. *read_rawimage* is an example of how to read a raw image into C. However, it is difficult to find a suitable file converter that converts other file formats to raw file
- It is found that older graphics cards cannot display texture property if the source file is not in $2^n$ x $2^m$