# CSCI 580 Discussion

## HW 5

*Lixing Liu*

*Slides modified from: Rongqi Qiu*

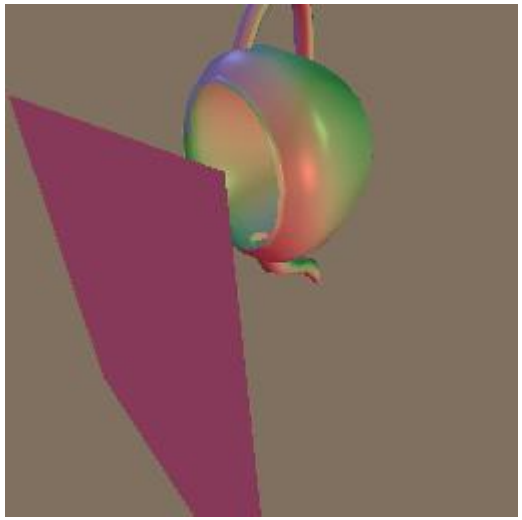
# HW4

- Due 10/9 11:59 pm
  No late submissions will be graded since HW4..

- Regrading for HW1 - 3
  - One-time regrading opportunity
  - Please write your requests in README.txt along with your HW4 submission

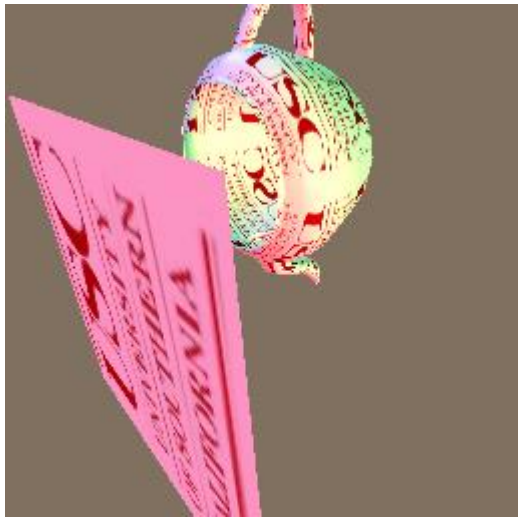
# HW5 – Overview

- Adding texturing capabilities to your renderer



HW4

Color specified by material attributes



HW5

Color specified by images

Color specified by functions

# HW5 – Overview

- Application5.cpp

Texture coordinates (u, v)

```
        /*
         * Set the value pointers to the first vertex of the
         * triangle, then feed it to the renderer
         * NOTE: this sequence matches the nameList token sequence
         */
        valueListTriangle[0] = (GzPointer)vertexList;
        valueListTriangle[1] = (GzPointer)normalList;
        valueListTriangle[2] = (GzPointer)uvList;
        GzPutTriangle(m_pRender, 3, nameListTriangle, valueListTriangle);
```

Texture function

```
        nameListShader[5]  = GZ_TEXTURE_MAP;
#if 0   /* set up null texture function or valid pointer */
        valueListShader[5] = (GzPointer)0;
#else
        valueListShader[5] = (GzPointer)(tex_fun);  /* or use ptex_fun */
#endif
```

# HW5 – Overview

- Application5.cpp

Texture coordinates (u, v)

Mapping from surface points to texture coordinates (u, v)

Texture function

- Image texture (tex_fun)

or

- Procedural texture (ptex_fun)

# HW5 – Outline

- Texture coordinates: surface point → (u, v)
  - Input: vertex → (u, v)
  - Replace (fixed) material color for (variable) texture color
- Texture functions: (u, v) → RGB
  - Image texture
    - 2D RGB image (look-up table)
  - Procedural texture
    - General computing functions

# HW5 – Outline

- Texture coordinates: surface point → (u, v)
  - Input: vertex → (u, v)
  - Replace (fixed) material color for (variable) texture color
- Texture functions: (u, v) → RGB
  - Image texture
    - 2D RGB image (look-up table)
  - Procedural texture
    - General computing functions

# HW5 – Texture Coordinates (GzPutTriangle)

- Assume we have defined a texture function
  - tex_fun(u, v) → RGB
- Two remaining problems:
  - How to apply texture at vertices of triangles?
  - How to apply texture at interior of triangles?

# HW5 – Texture Coordinates (GzPutTriangle)

- Assume we have defined a texture function
  - tex_fun(u, v) → RGB
- Two remaining problems:
  - **How to apply texture at vertices of triangles?**
  - How to apply texture at interior of triangles?

# HW5 – Texture Coordinates (GzPutTriangle)

- How to apply texture at vertices?
  - Retrieve (u, v) at this vertex (from input)
  - Call tex_fun to retrieve RGB color
  - What to do with retrieved RGB color?
    - Phong shading: Ka, Kd
    - Gouraud shading: Ks, Ka, Kd

# HW5 – Texture Coordinates (GzPutTriangle)

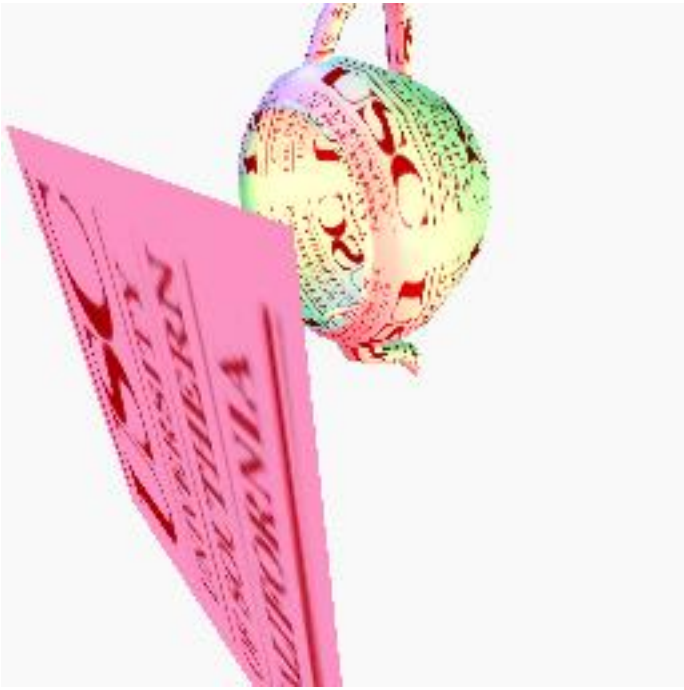
- Assume we have defined a texture function
  - tex_fun(u, v) → RGB
- Two remaining problems:
  - How to apply texture at vertices of triangles?
  - **How to apply texture at interior of triangles?**

# HW5 – Texture Coordinates (GzPutTriangle)

- Texture coordinates (u, v) are given only at vertices, but interior points need (u, v) too.
- Solution: interpolation!
  - Similar to interpolating **Z** (during rasterization), **RGB color** (during Gouraud shading) and **normal vectors** (during Phong shading).
  - Either LEE or scan-line is fine.
  - Note: perspective correction (Class7, Slide 14)

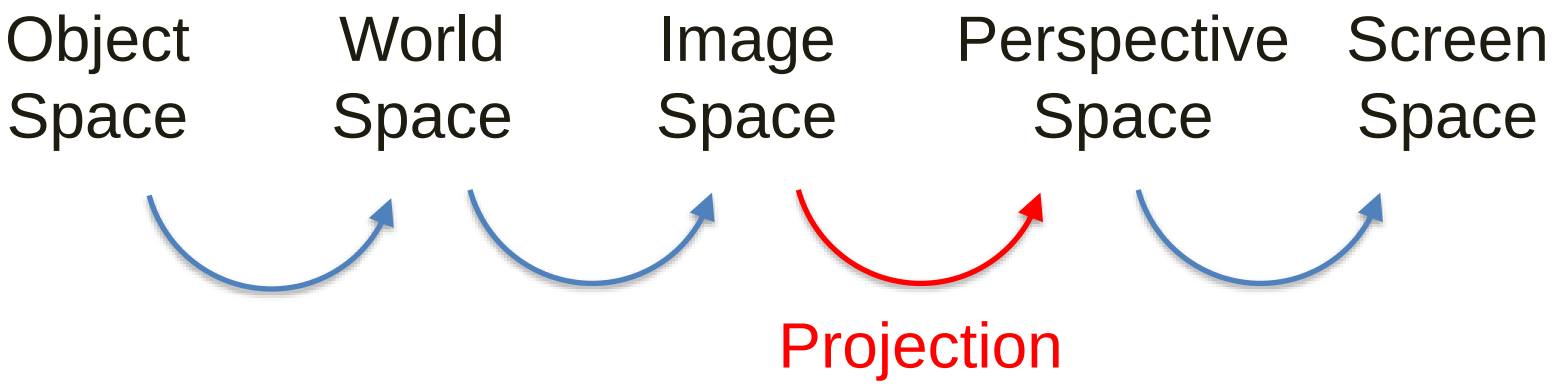
# HW5 – Perspective Correction

perspective correction

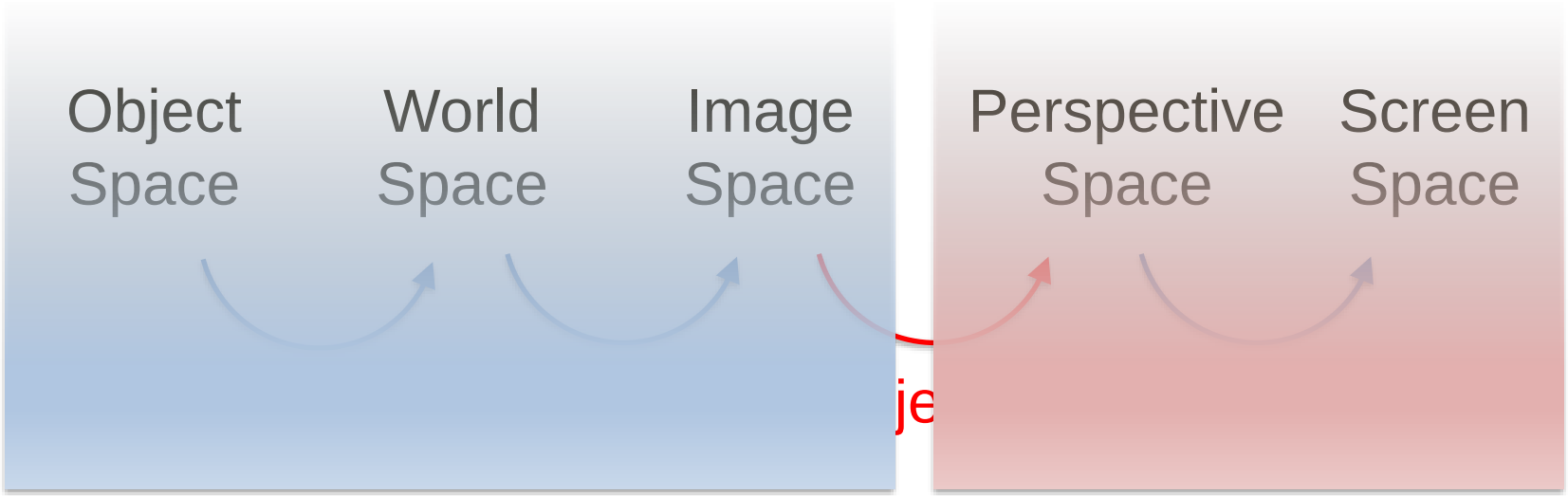


no perspective correction

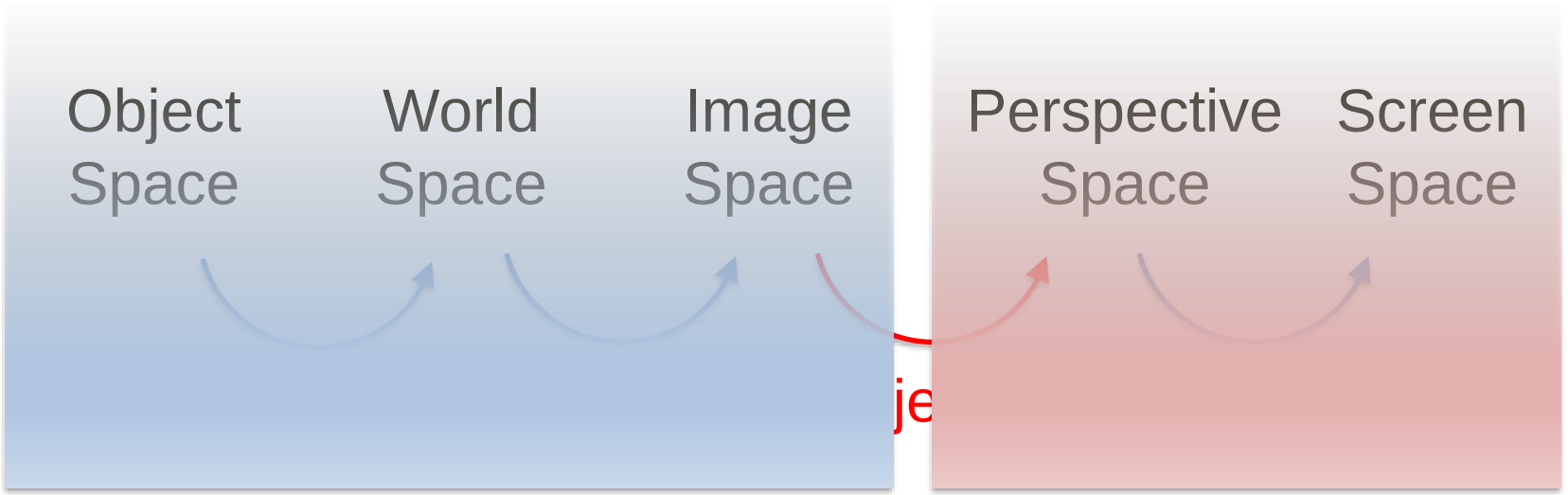
# HW5 – Perspective Correction



# HW5 – Perspective Correction



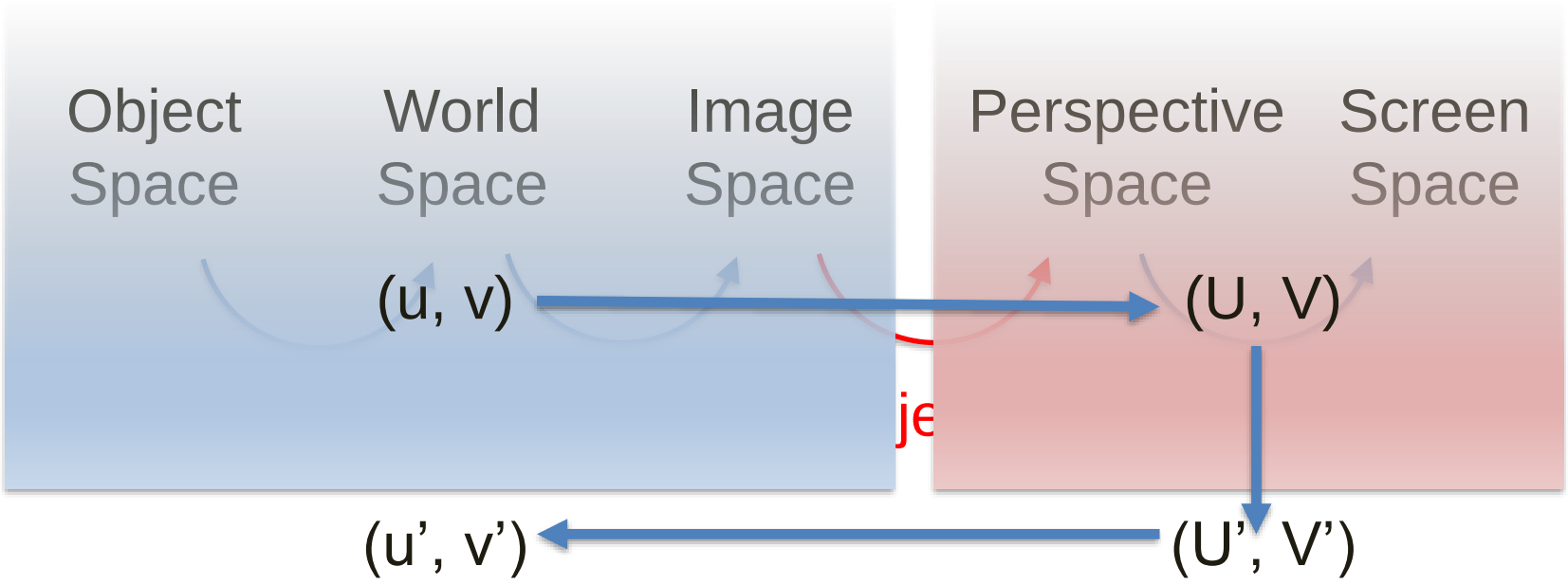
# HW5 – Perspective Correction




This is where (u, v) is given.

This is where pixels are drawn.

# HW5 – Perspective Correction



# HW5 – Texture Coordinates (GzPutTriangle)

- Phong shading

  $C = (K_s \sum_L [I_e (R \bullet E)^s] ) + (K_d \sum_L [I_e (N \bullet L)]) + (K_a I_a)$

material attribute (fixed)  texture lookup (pixel-by-pixel)

- Gouraud shading (set $K_T = K_s = K_d = K_a$)

  $C = (K_T) (\sum_L [I_e (R \bullet E)^s] + \sum_L [I_e (N \bullet L)] + I_a)$

texture lookup (pixel-by-pixel)  compute at vertices and interpolate to pixels

# HW5 – Outline

- Texture coordinates: vertex -> (u, v)
  - Given as input per vertex
  - Used for obtaining (variable) texture color in replacement of (fixed) material color
- Texture functions: (u, v) -> RGB
  - Image texture
    - 2D RGB image (look-up table)
  - Procedural texture
    - General computing functions

# HW5 – Image texture (tex_fun)

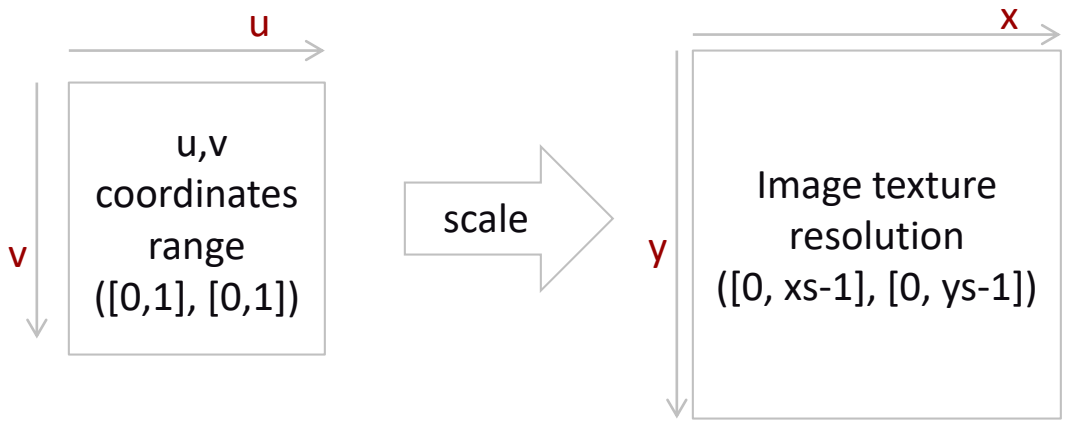
- Compute RGB color given texture coordinates (u, v)

```
/* Image texture function */
int tex_fun(float u, float v, GzColor color)
```

- Returns GZ_SUCCESS or GZ_FAILURE
- Boundary test (u,v) within [0,1] x [0,1]
- The codes for loading texture files are already given

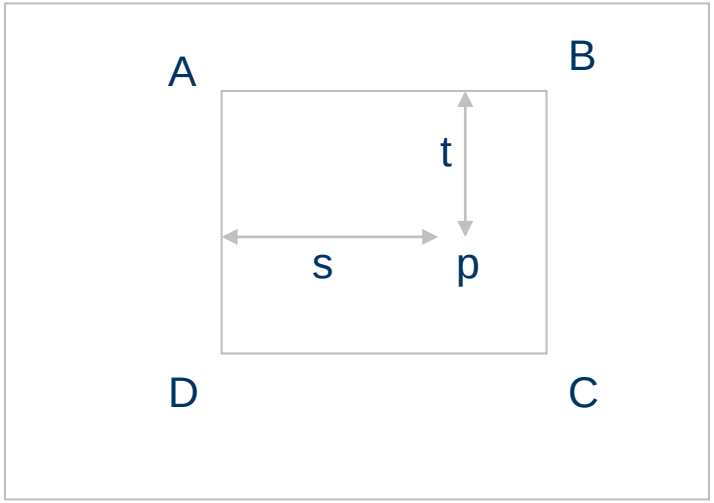
# HW5 – Image texture (tex_fun)

- Problem: (u, v) range over [0, 1] x [0, 1],
  texture image is 2D array of (xs-1) x (ys-1)
- Solution: scaling by the size of the texture image



# HW5 – Image texture (tex_fun)

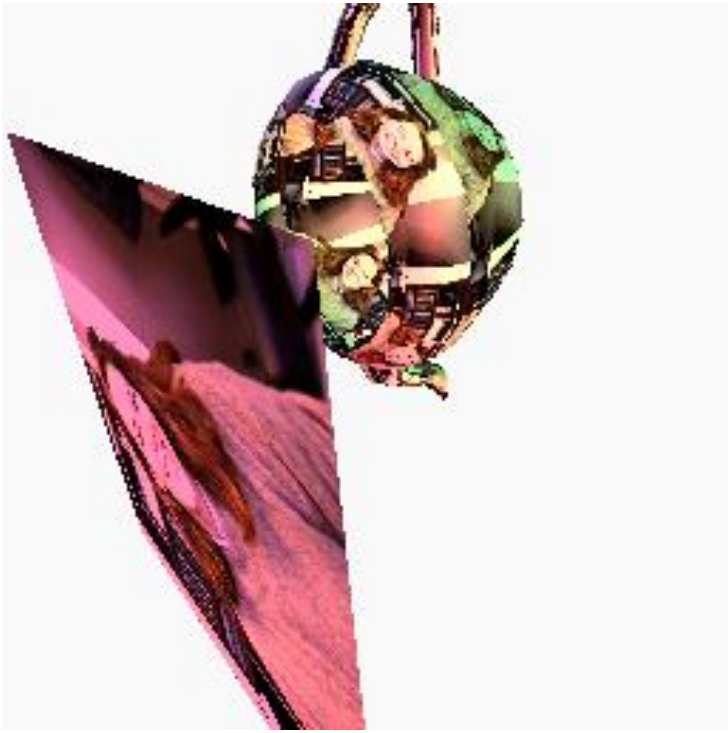
- Problem: what if scaled (u, v) is not integer?
- Solution: bilinear interpolation
- Color(p) = s t C + (1-s) t D + s (1-t) B + (1-s) (1-t) A
  - s, t are fractional distances [0,1]
  - A, B, C, D are RGB colors at neighboring pixels



# HW5 – Image texture (tex_fun)

- Try other texture images!
- Just replace the file “texture” with any ppm image.

Example texture image



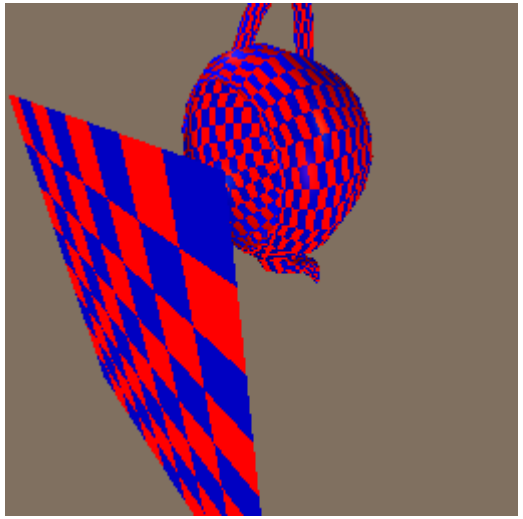
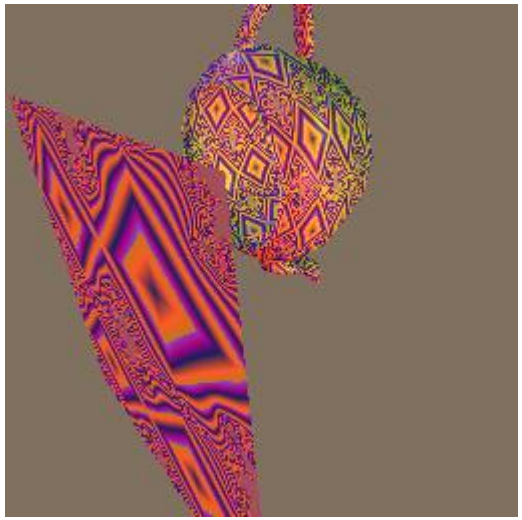
Rendering using the left texture image

# HW5 – Outline

- Texture coordinates: vertex -> (u, v)
  - Given as input per vertex
  - Used for obtaining (variable) texture color in replacement of (fixed) material color
- Texture functions: (u, v) -> RGB
  - Image texture
    - 2D RGB image (look-up table)
  - Procedural texture
    - General computing functions

# HW5 – Procedural texture (ptex_fun)

- Any function of (u,v) is OK as long as a texture pattern is clearly evident. (i.e. Julia set)
- Use your imagination!


# HW5 pitfalls

- - Perspective-Z: make sure the texture is not distorted on the plane and the teapot.
- - Bilinear interpolation: make sure the texture is not too aliased.
- - Procedural texture: do not forget to implement a procedural texture