Fundamentals of Computer Graphics

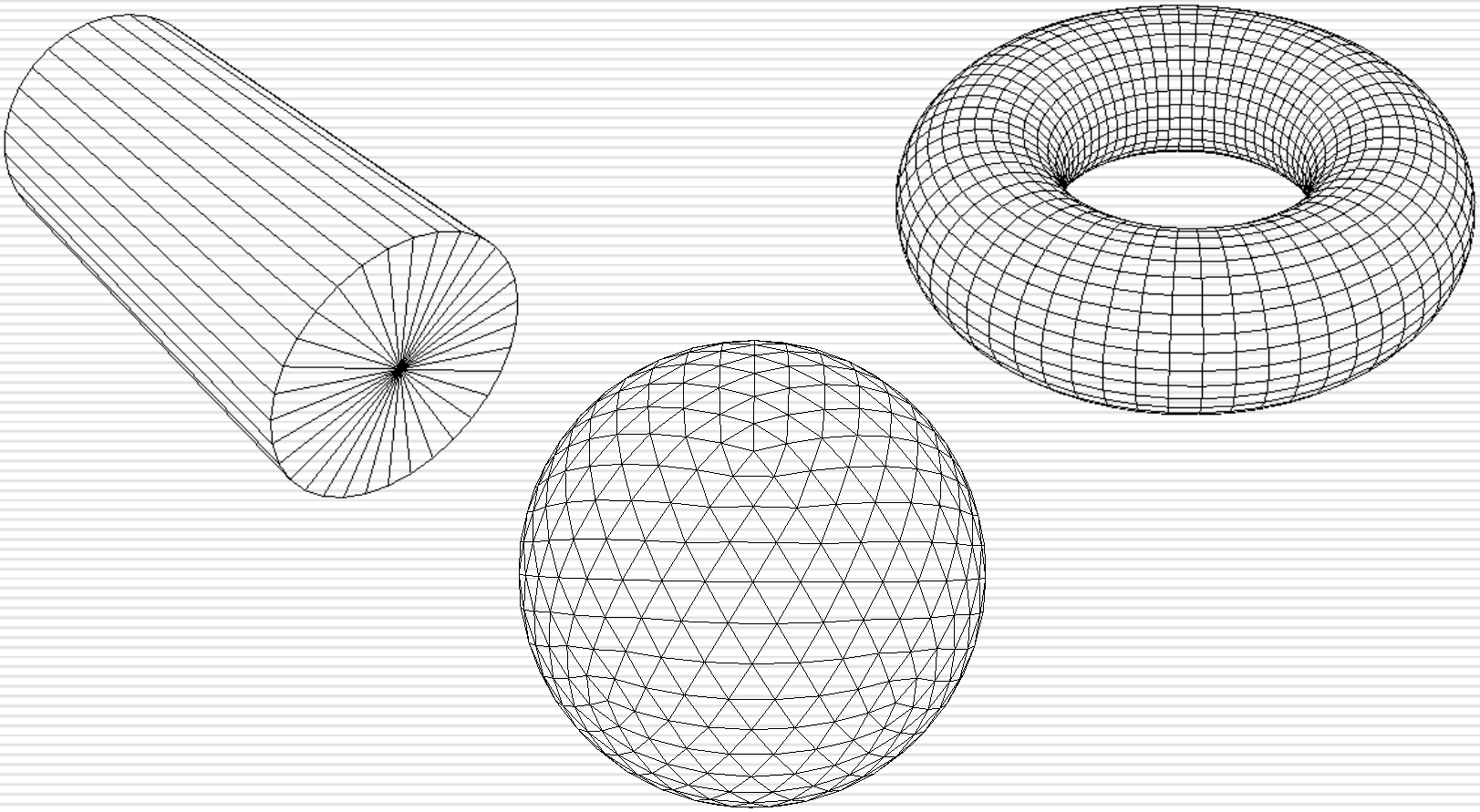
Lecture 5. Texture Mapping

Yong-Jin Liu
liuyongjin@tsinghua.edu.cn

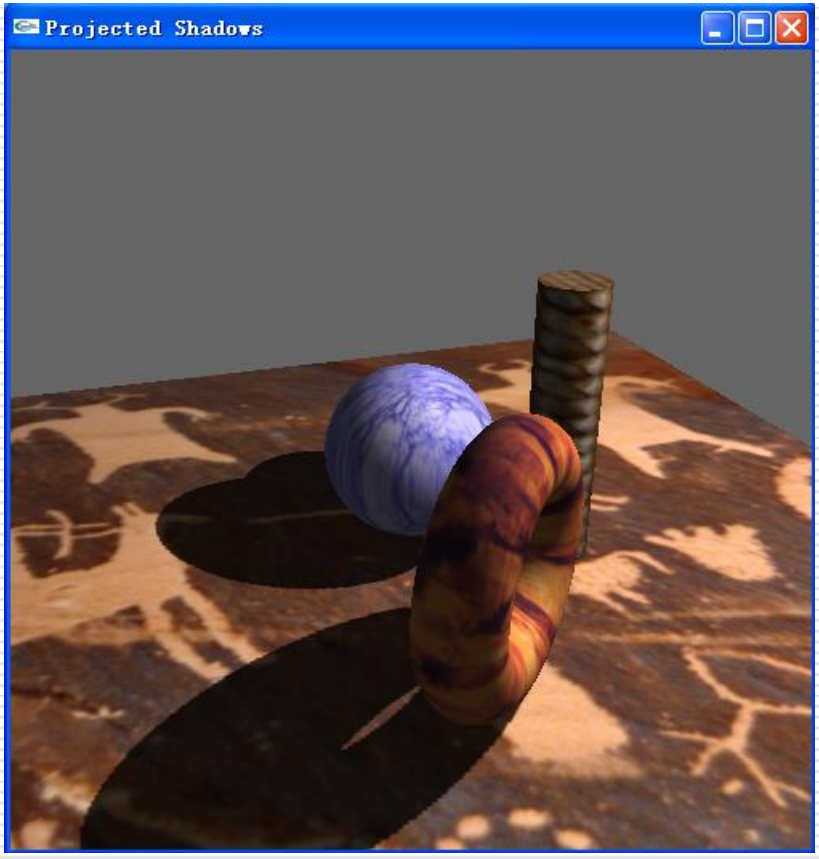
Next week course

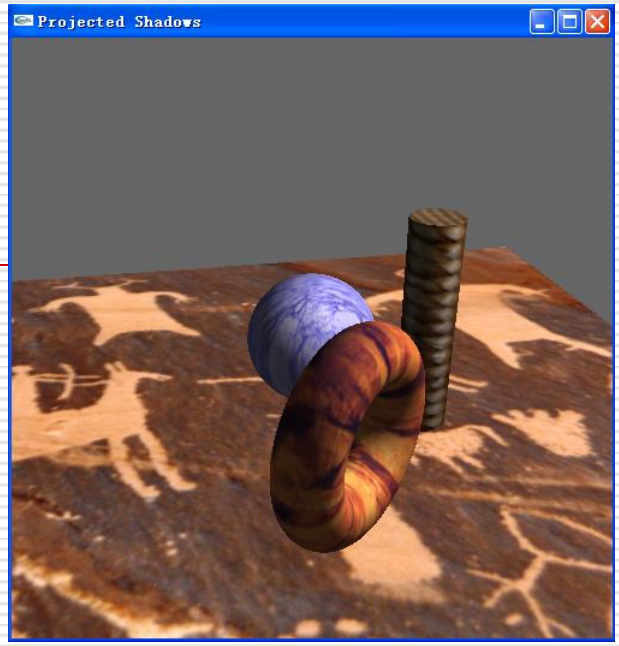
- Monday (4th April, public holiday)
- Move to Tuesday (5th April)

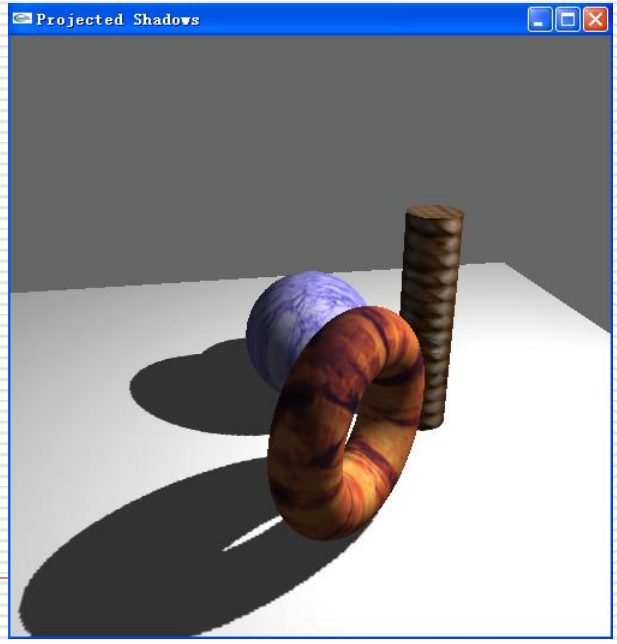
Procedure modeling



Texture mapping







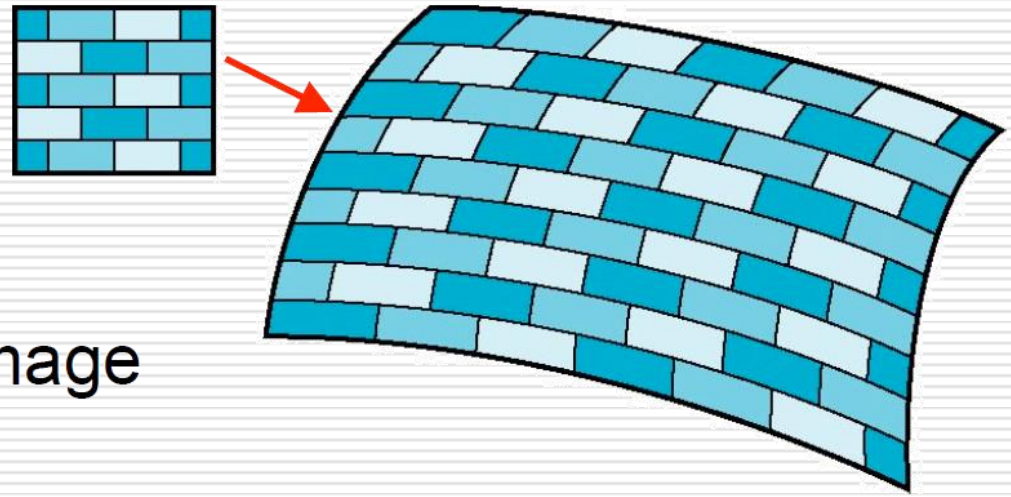
Outline

- What is texture mapping?
- OpenGL texture functions
- Texture mapping applications

What is texture mapping?

Texture mapping

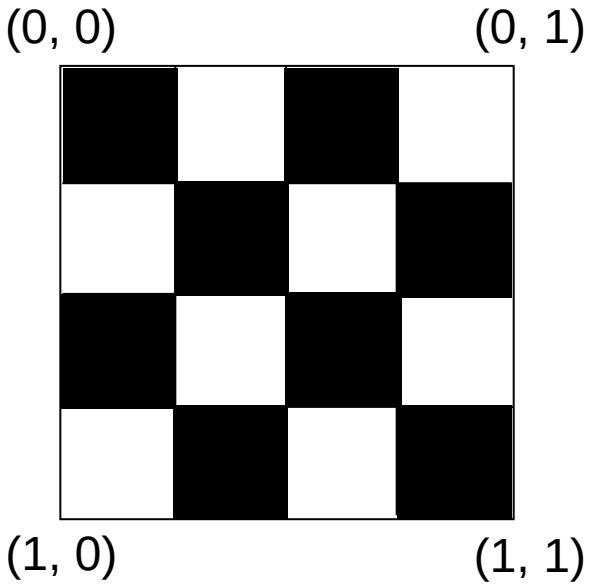
- The idea is simple ---- map an image to a surface

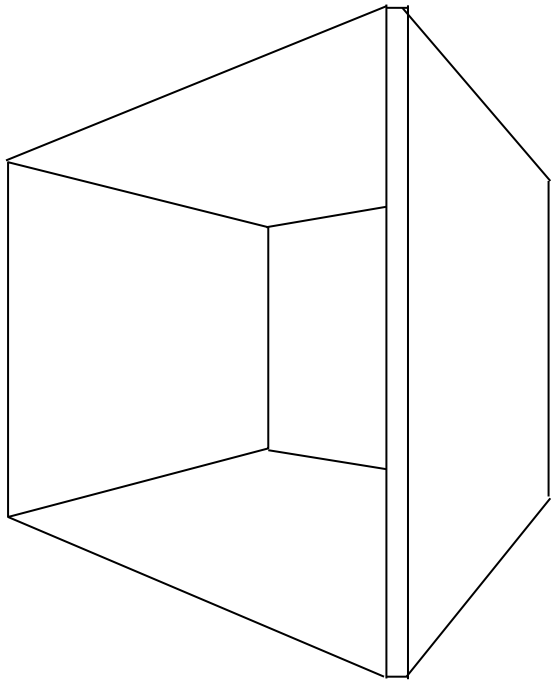


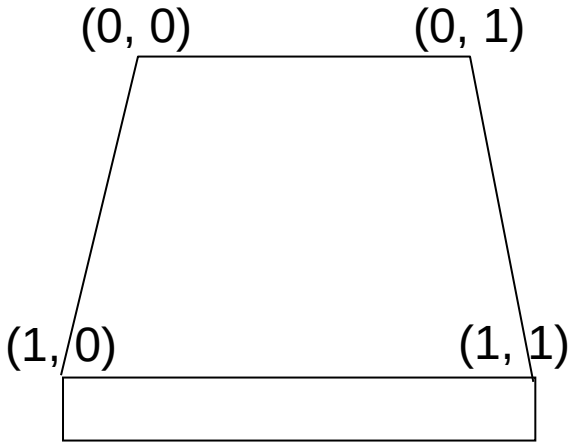
What is texture mapping?

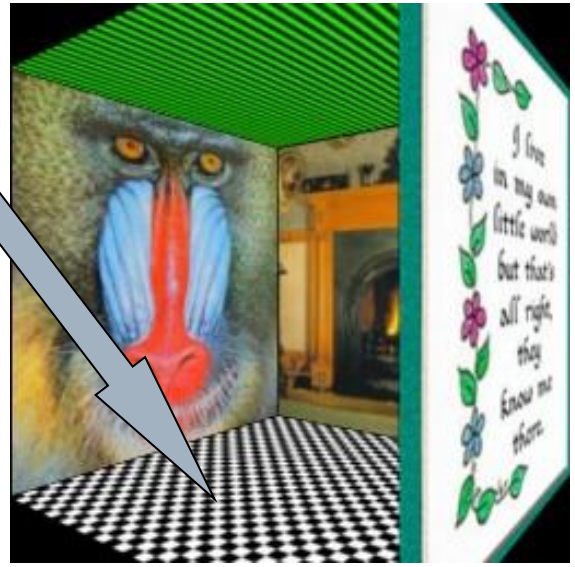
Texture mapping

- Allows you to glue an image of a brick wall (e.g., an image/photograph) to a polygon and to draw the entire wall as a single polygon.
- Ensure that all the right things happen as the polygon is transformed and rendered.

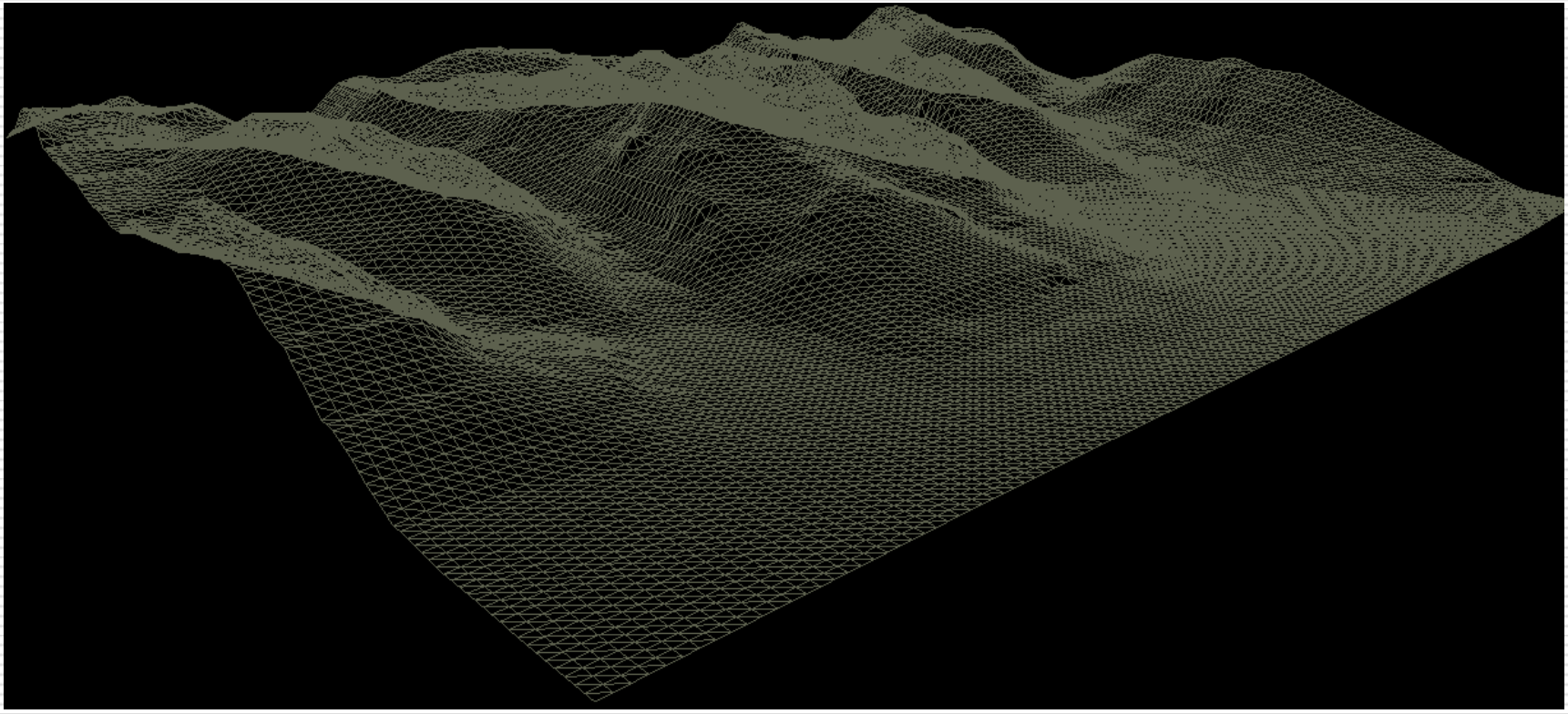

(0, 0)
(0, 1)
(1, 0)
(1, 1)



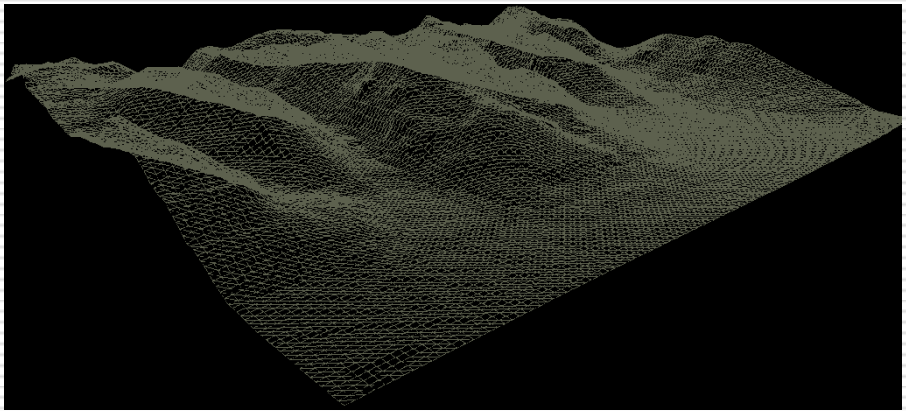

(0, 0)
(0, 1)
(1, 0)
(1, 1)

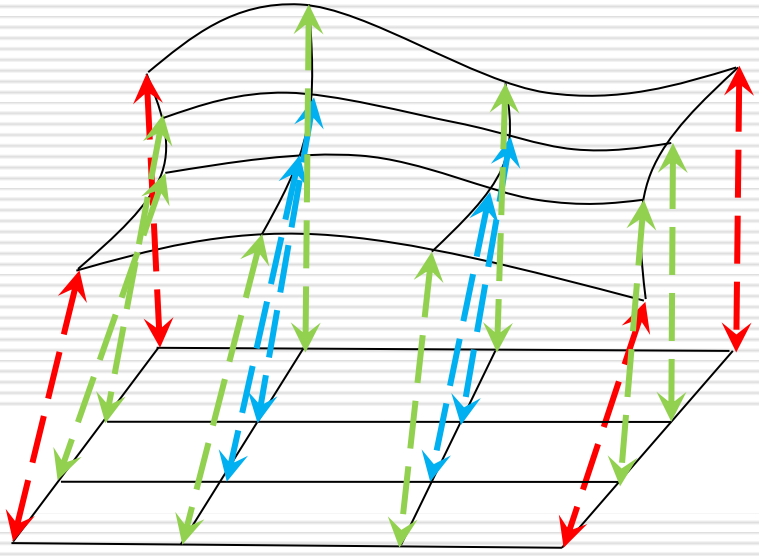


An example



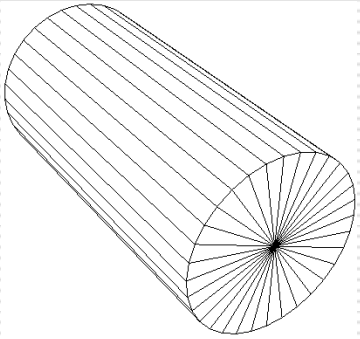
An example

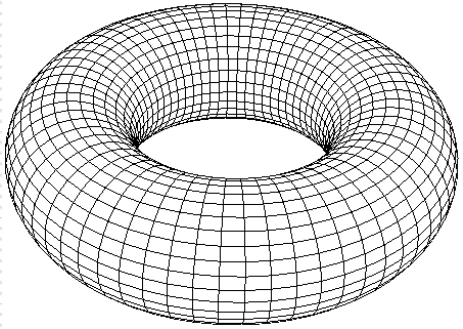


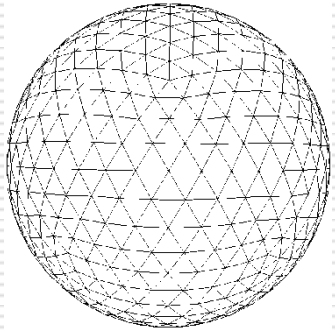


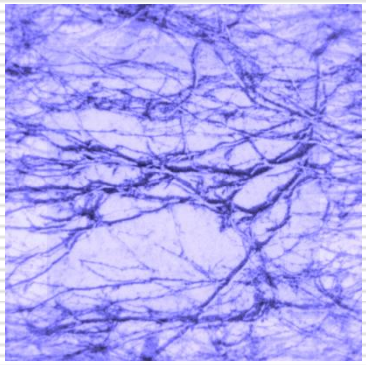
An example

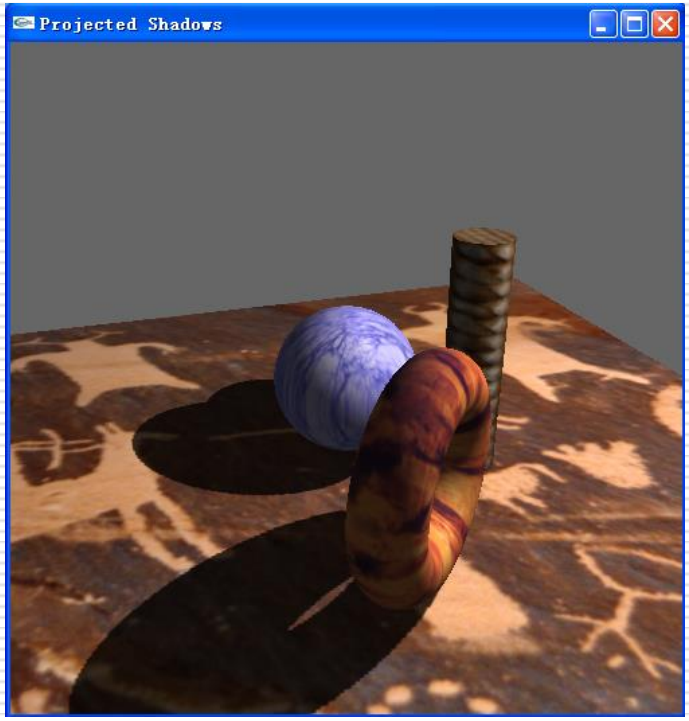
More examples



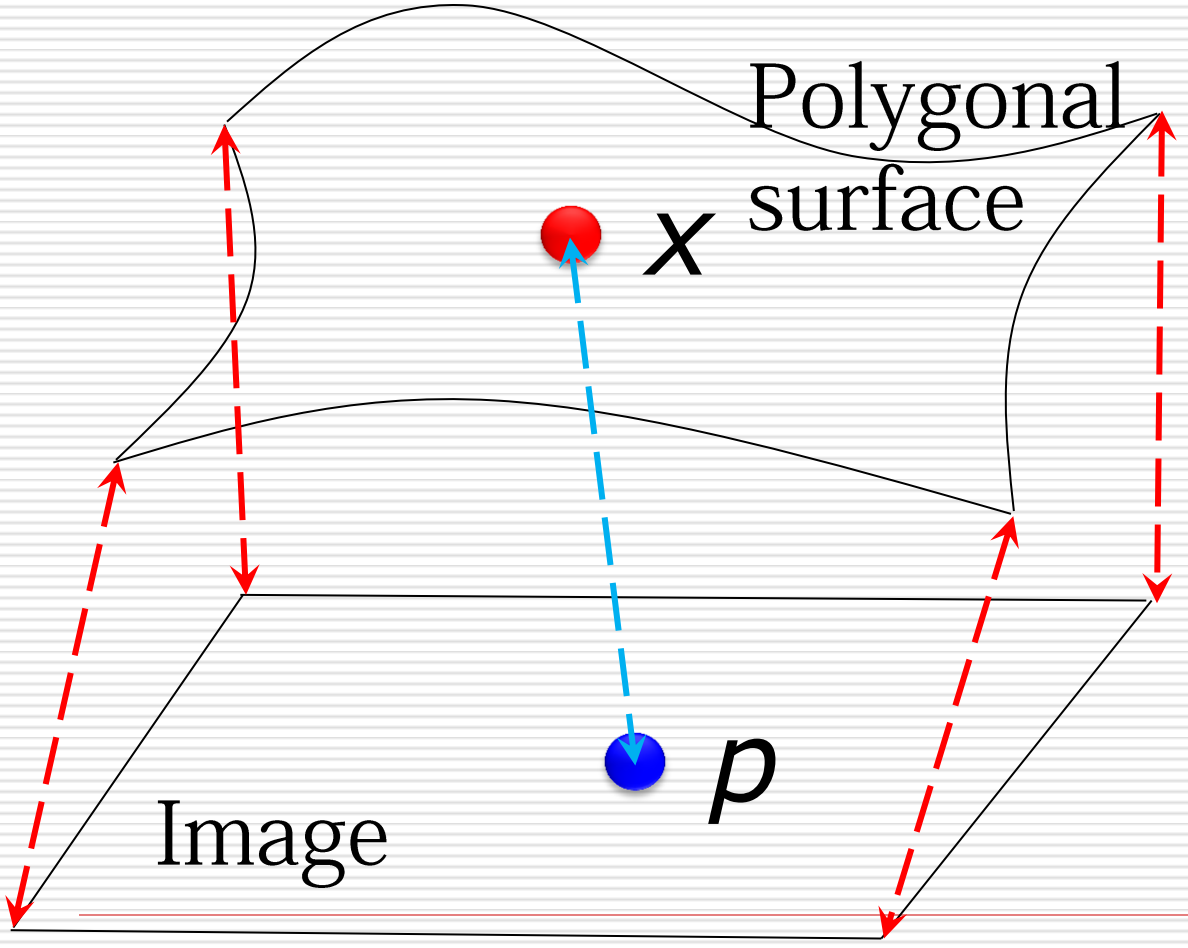








Basic idea



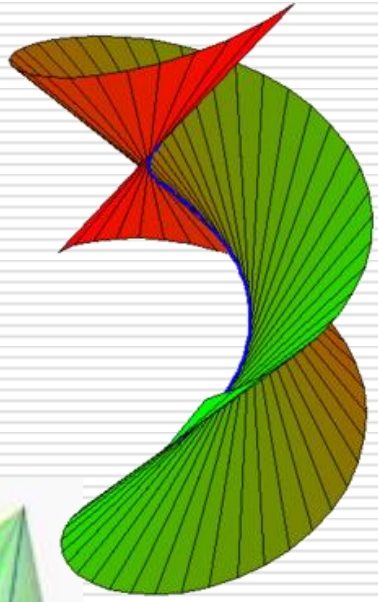
$$f(x) = p$$

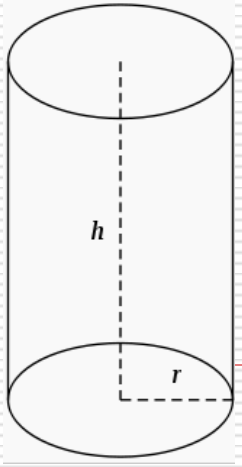
Developable surfaces

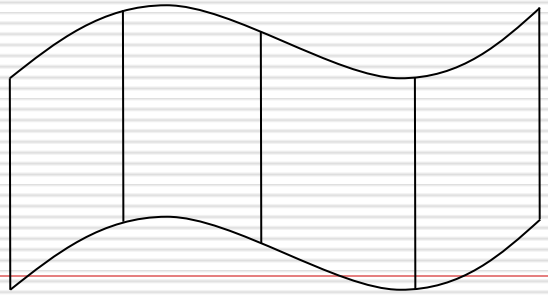
- Ideally mapping:
  a "surface" is flattened onto a plane without distortion (i.e. without "stretching" or "compressing").
- Developable surfaces
  a surface with zero Gaussian curvature

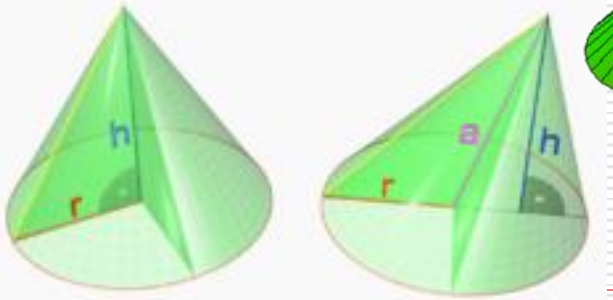
Developable surfaces

- Only four types of developable surfaces
  - Plane
  - Cylinder
  - Cone
  - Tangent surface of a space curve



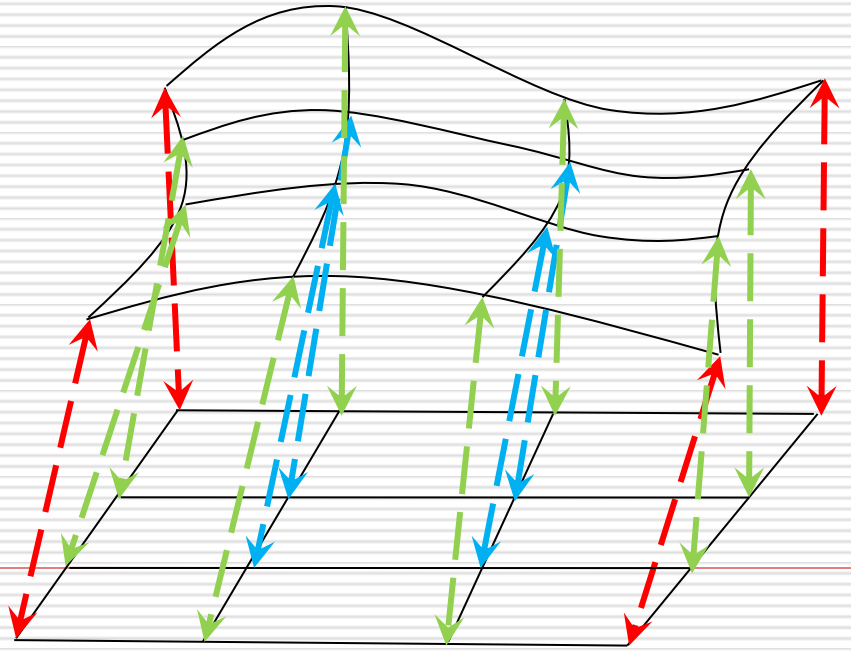






Most surfaces are undevelopable

- Folding into a plane will lead to distortion ("stretching" or "compressing") somewhere inevitably



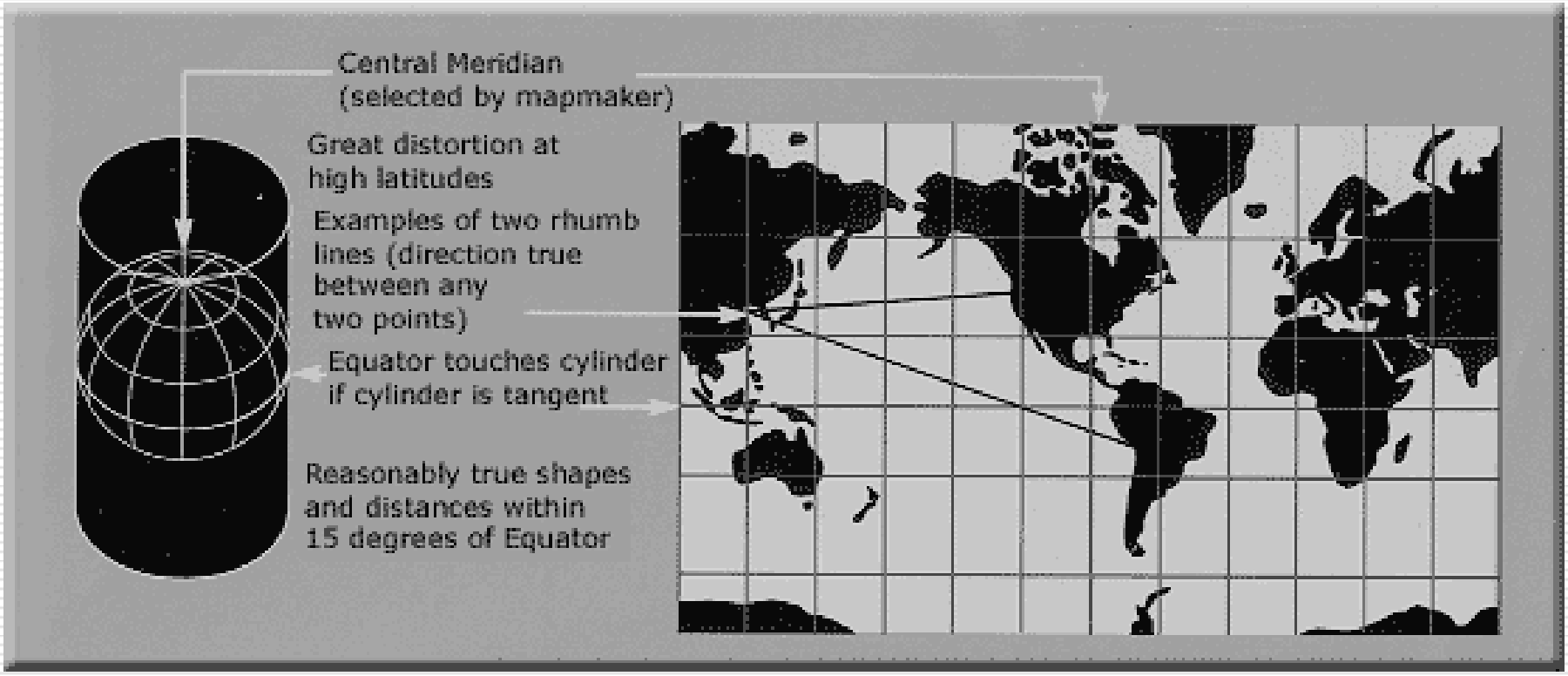
Most surfaces are undevelopable

- Different optimization criteria
  - Length distortion minimization
  - Area distortion minimization
  - Angle distortion minimization
  - … … …
- Applications

Cartography

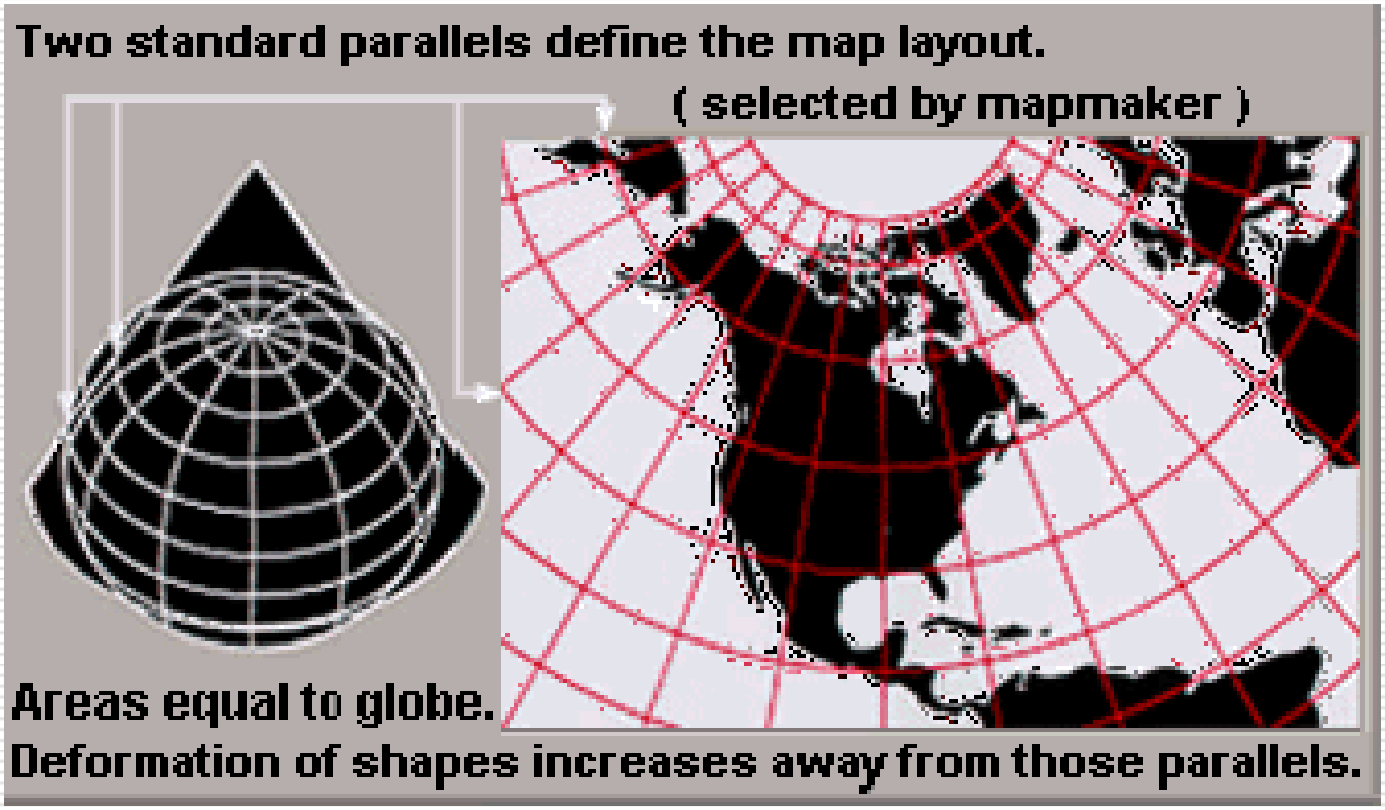
Draftsmanship

Central Meridian （子午线投影）



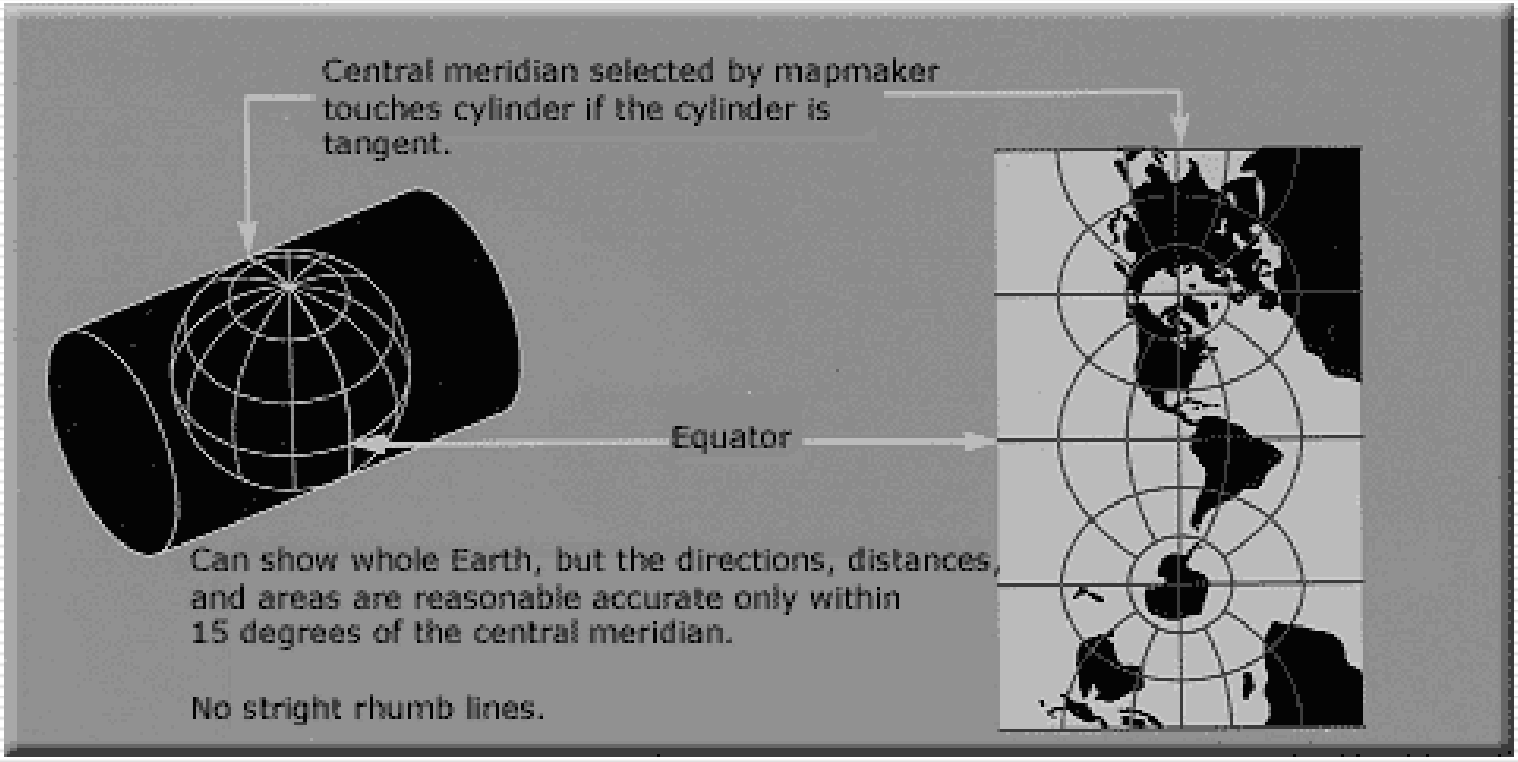
Near equator is better; near polar region is worse

Albers Projection

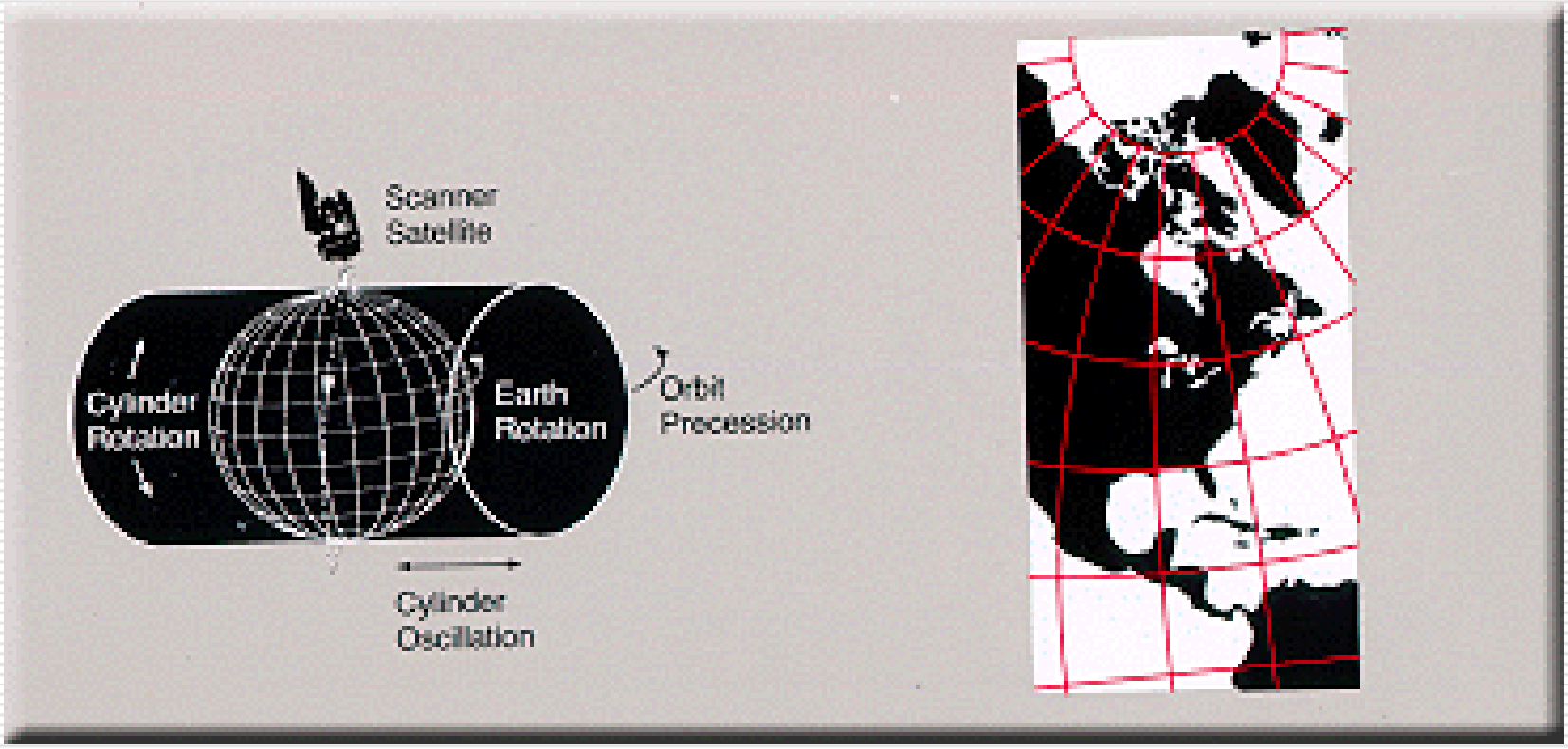


Area preserving is good; but shape distortion is large

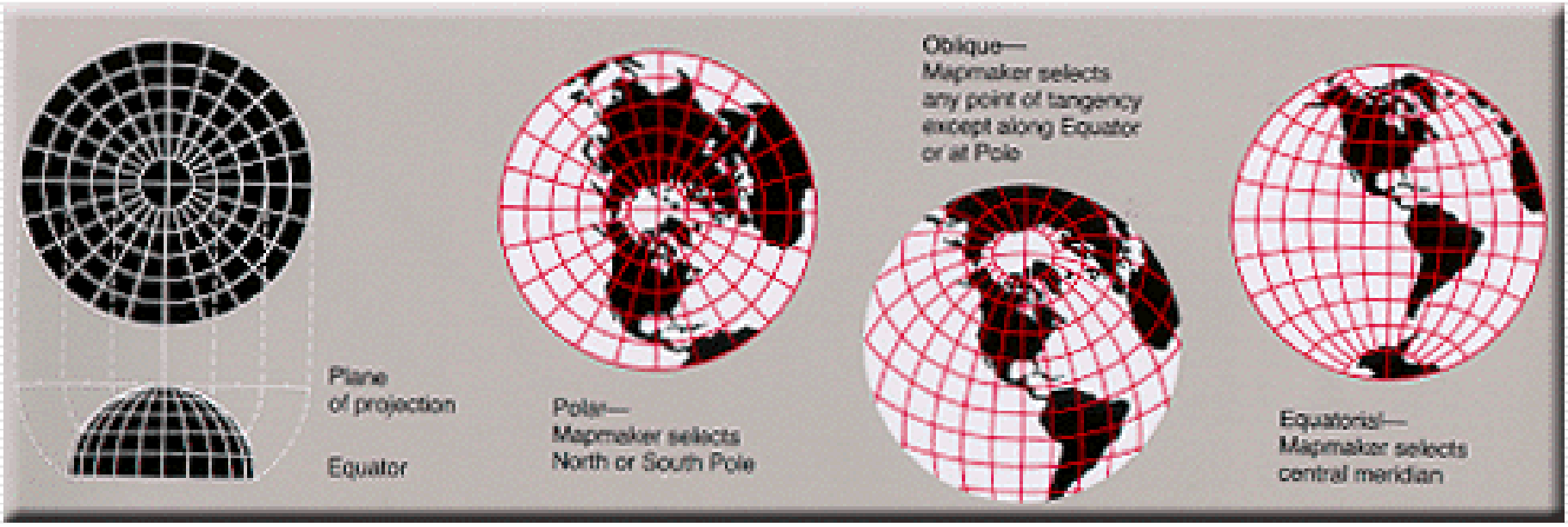
Transverse Meridian Projection



Space-oblique Mercator projection

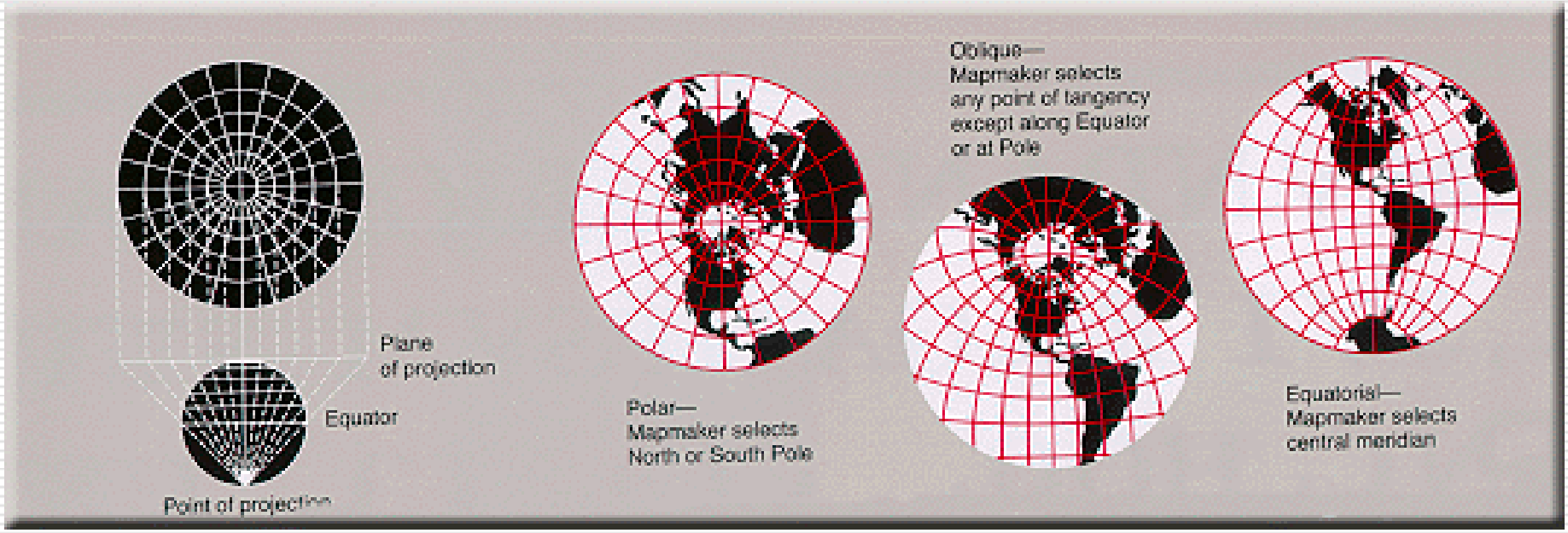


Azimuthal Projection



Directions from center are good; but shape and Length changes largely

Stereographic Projection



Angle preserving; but not area and length

OpenGL texture mapping

Three steps to applying a texture

1. Specify the texture image
2. Assign texture coordinates to vertices
3. Specify texture parameters

OpenGL texture mapping

Three steps to applying a texture

1. Specify the texture image
   - Read or generate image
   - Assign to texture
   - Enable texturing
2. Assign texture coordinates to vertices
3. Specify texture parameters

OpenGL texture mapping

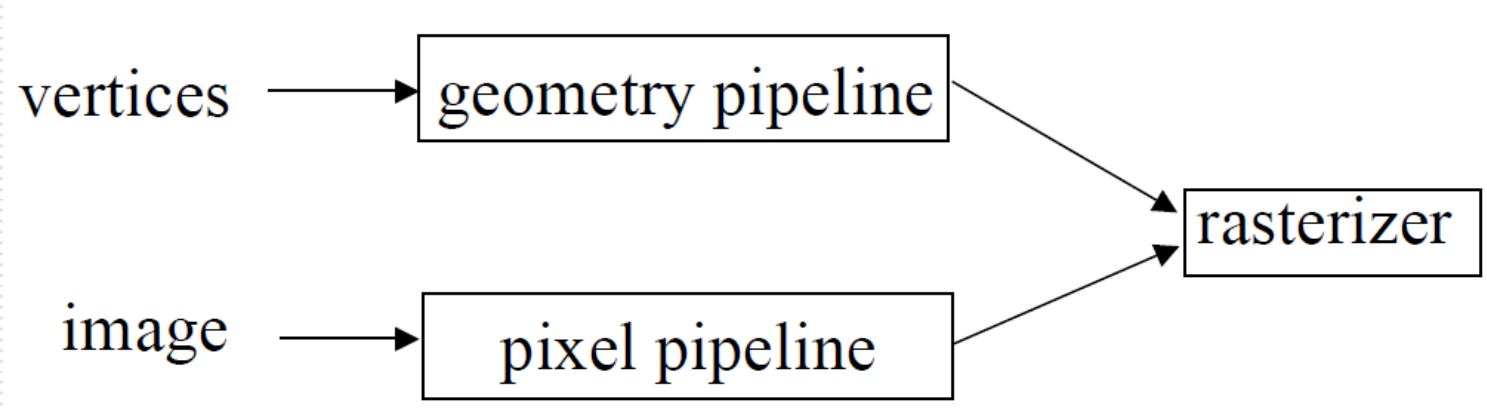
Three steps to applying a texture

1. Specify the texture
2. Assign texture coordinates to vertices
   - Suitable mapping function is left to application
3. Specify texture parameters
   - Wrapping, filtering

OpenGL rendering pipeline

Images and geometry flow through separate pipelines that join at the rasterizer

- Complex textures do not affect geometric complexity.

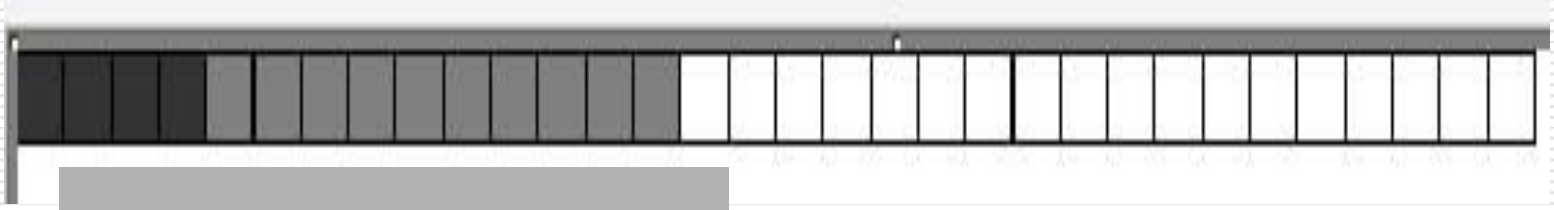


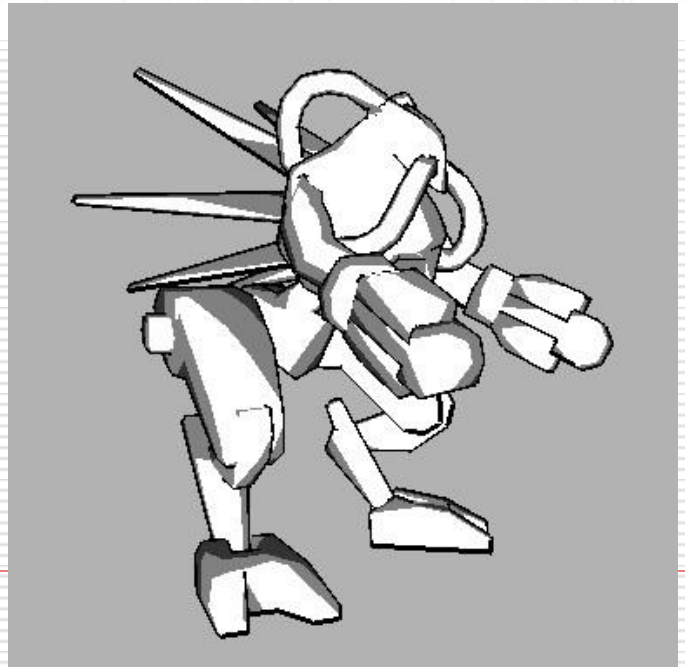
Step 1.1: specifying a texture image

- Define a texture image from an array of texels (texture elements) in CPU memory
  - Glubyte my_textels[512][512][3];
- 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

Quick overview

1D texture, its pixels are only white, gray of different scale and black





Cartoon style shading
（1D texture）

Quick overview

3D texture: solid, not only on the surface

Step 1.2: 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);
```

Step 1.3: Convert a texture image

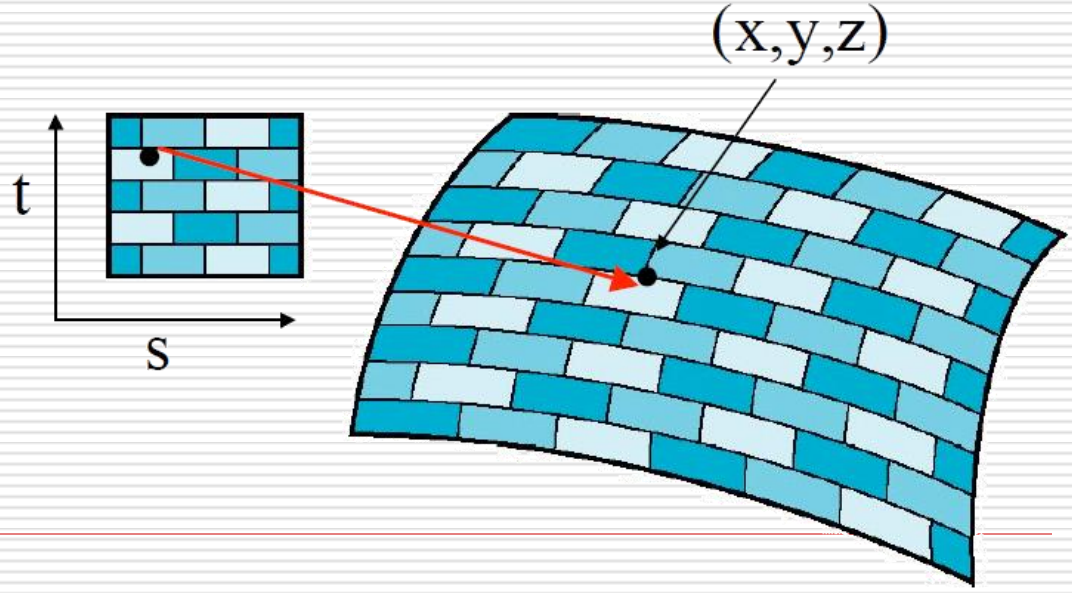
- OpenGL requires texture dimensions to be powers of 2
- If dimensions of image are not powers of 2

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

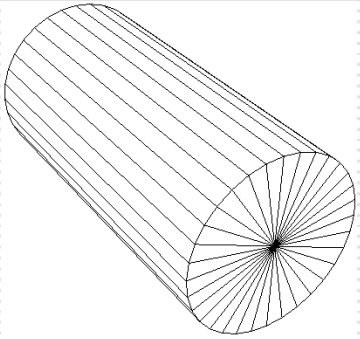
  - `data_in` is source image
  - `data_out` is for destination image
- Image interpolated and filtered during scaling

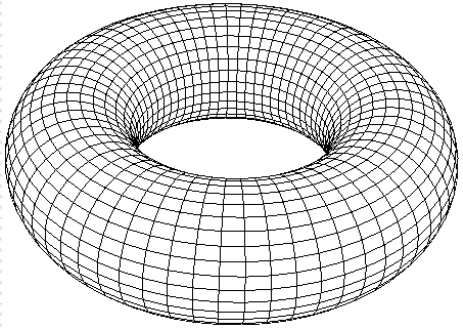
OpenGL texture mapping

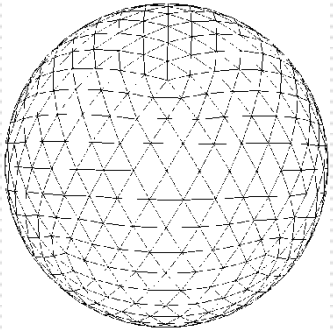
1. Specify the texture
2. Assign texture coordinates to vertices
3. Specify texture parameters

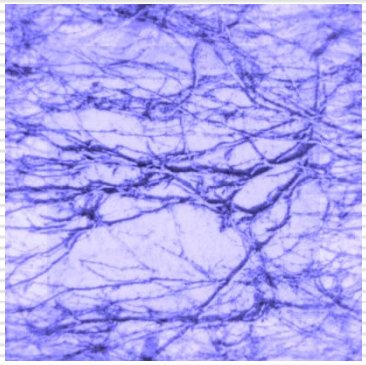


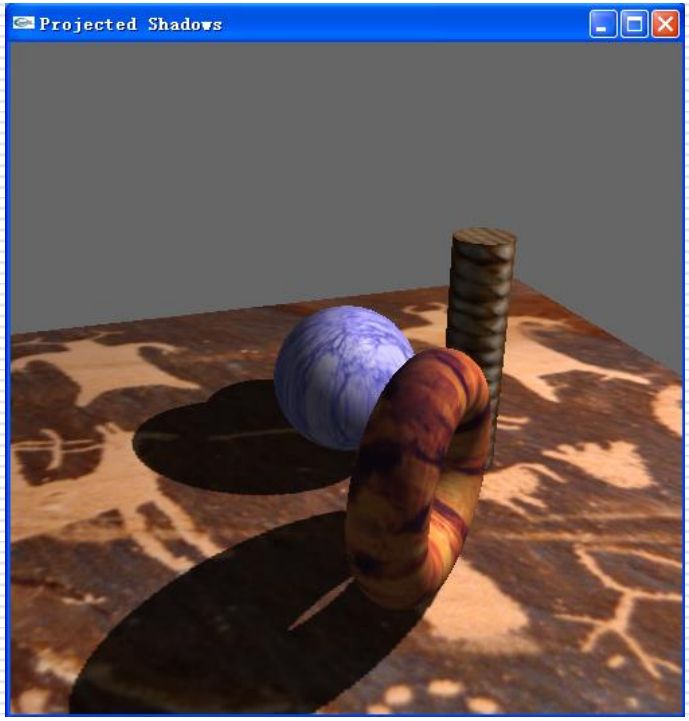
Using parametric domain







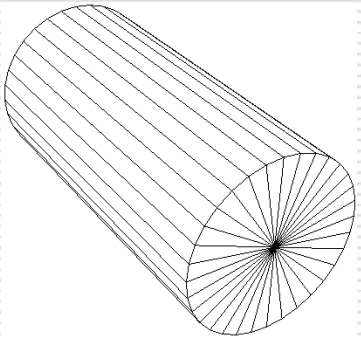




Using parametric domain

parametric 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 coordinates

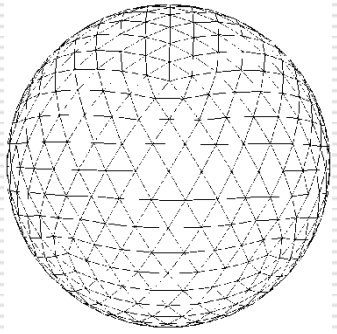
$$s = u$$
$$t = v$$

maps from texture space

Using parametric domain

We can use a parametric sphere

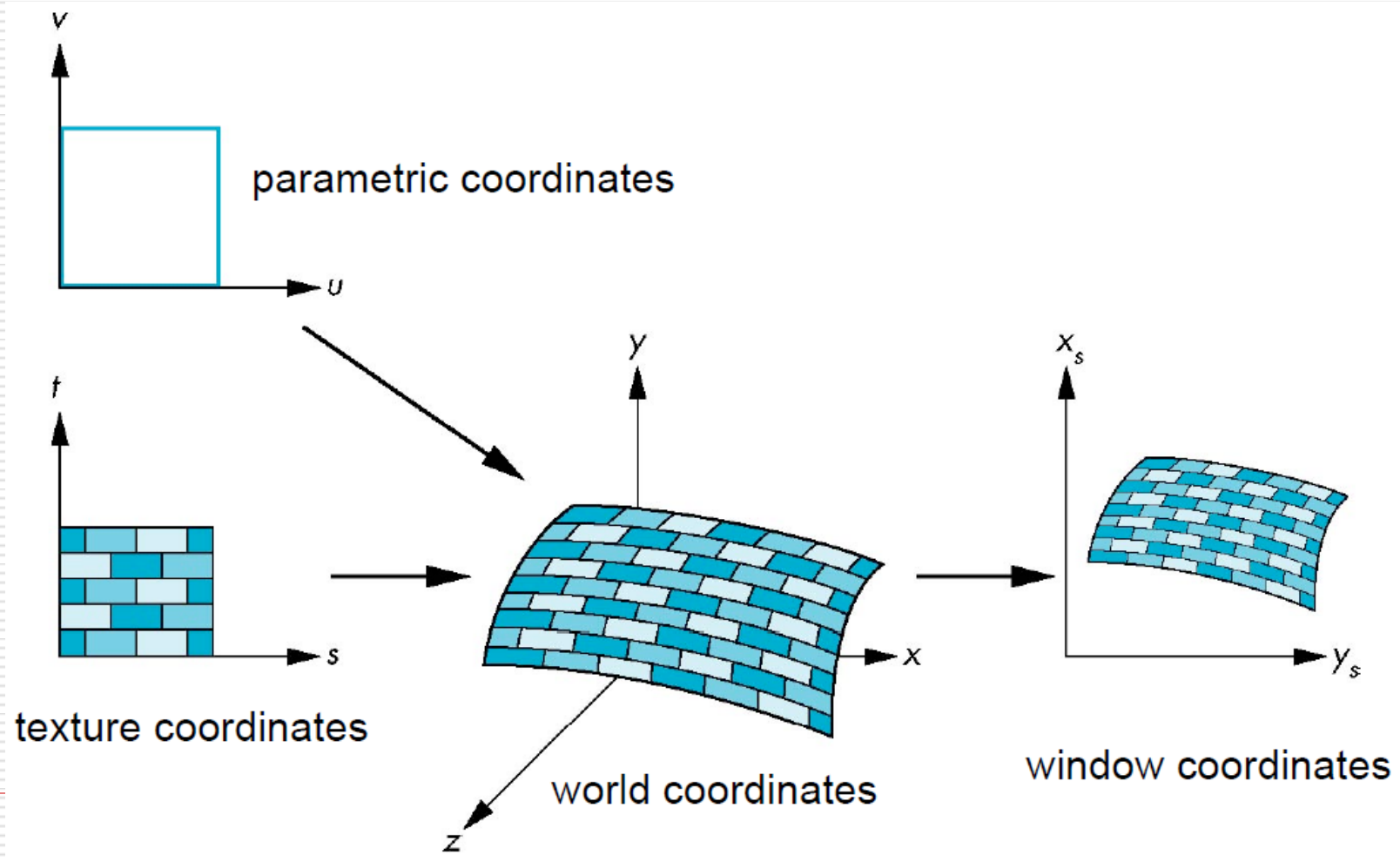
$$x = r \cos 2\pi u$$
$$y = r \sin 2\pi u \cos 2\pi v$$
$$z = r \sin 2\pi u \sin 2\pi v$$



in a similar manner to the cylinder but have to decide where to put the distortion

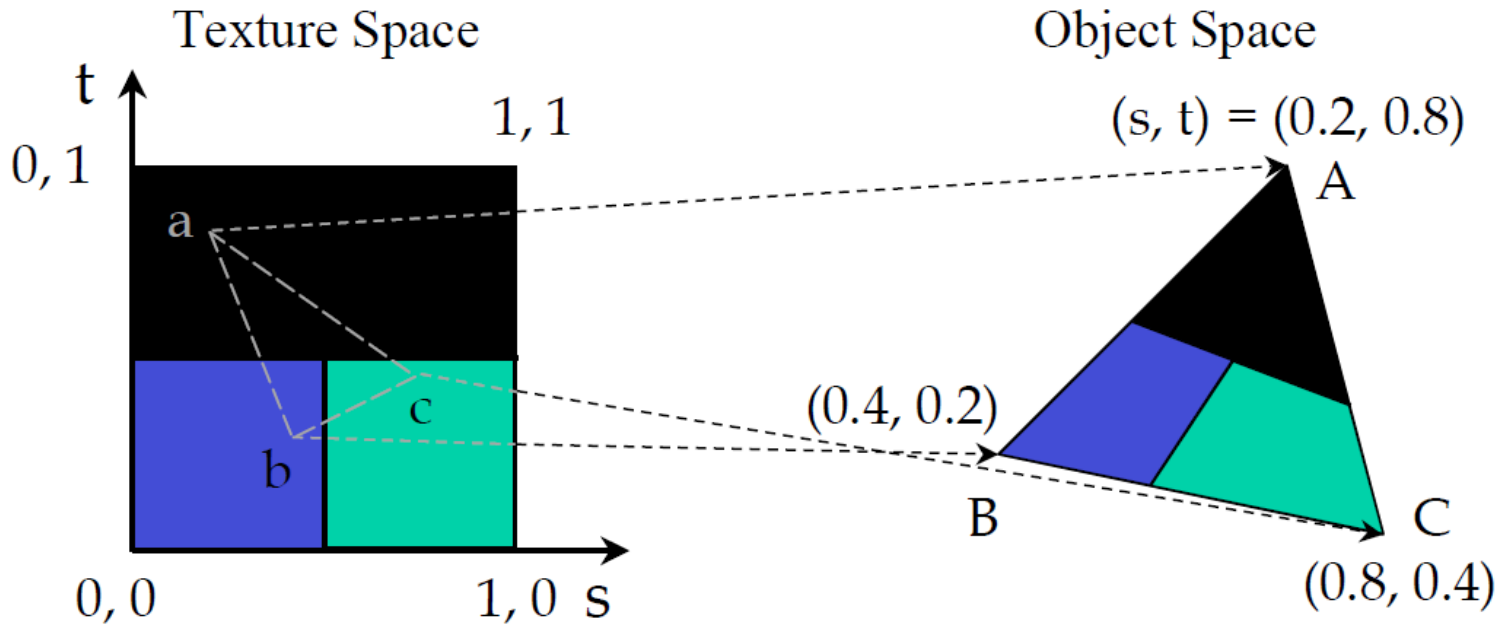
Spheres are used in environmental maps

Using parametric domain



Step 2: 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();
```

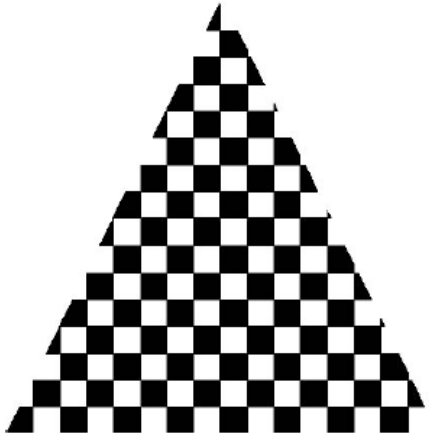
Note that we can use vertex arrays to increase efficiency

Interpolation

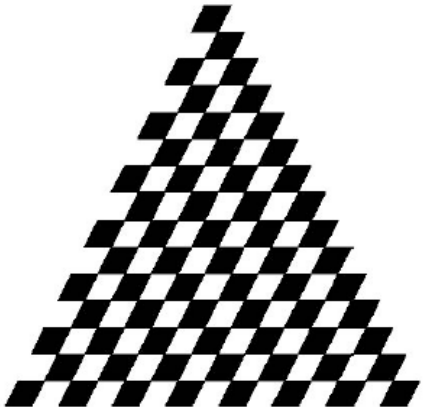
OpenGL uses interpolation to find proper texels from specified texture coordinates

Can be distortions

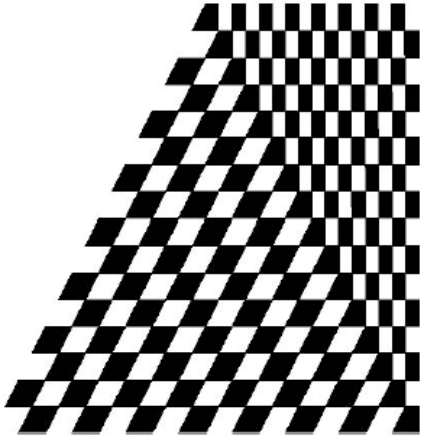
good selection of tex coordinates



poor selection of tex coordinates



texture stretched over trapezoid showing effects of bilinear interpolation



Step 3. Texture parameters

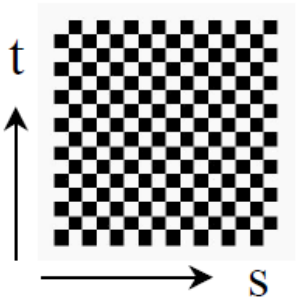
- OpenGL has a variety of parameters that determine how texture is applied
  - Wrapping parameters determine what happens if s and t are outside the (0,1) range
  - Filter modes allow us to use area averaging instead of point samples
  - Mipmapping allows us to use textures at multiple resolutions
  - Environment parameters determine how texture mapping interacts with shading

Warping mode

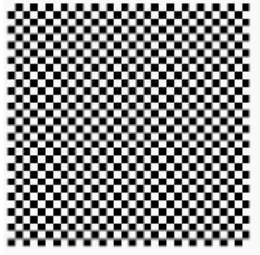
Clamping: if s,t > 1 use 1, if s,t <0 use 0

Wrapping: use s,t modulo 1

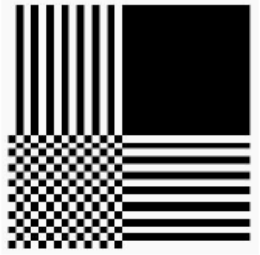
```
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_S, GL_CLAMP )
glTexParameteri( GL_TEXTURE_2D,
      GL_TEXTURE_WRAP_T, GL_REPEAT )
```



texture

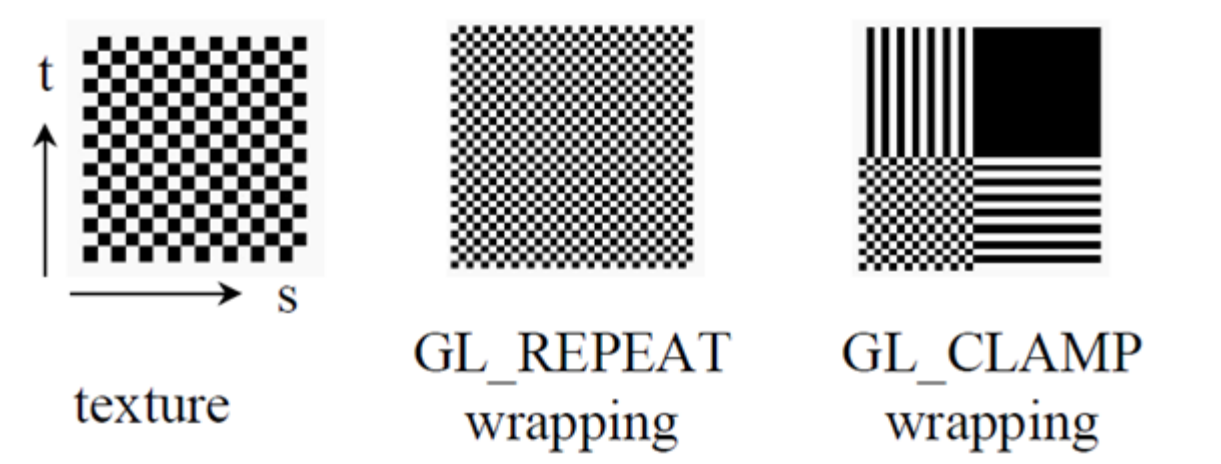


GL_REPEAT wrapping



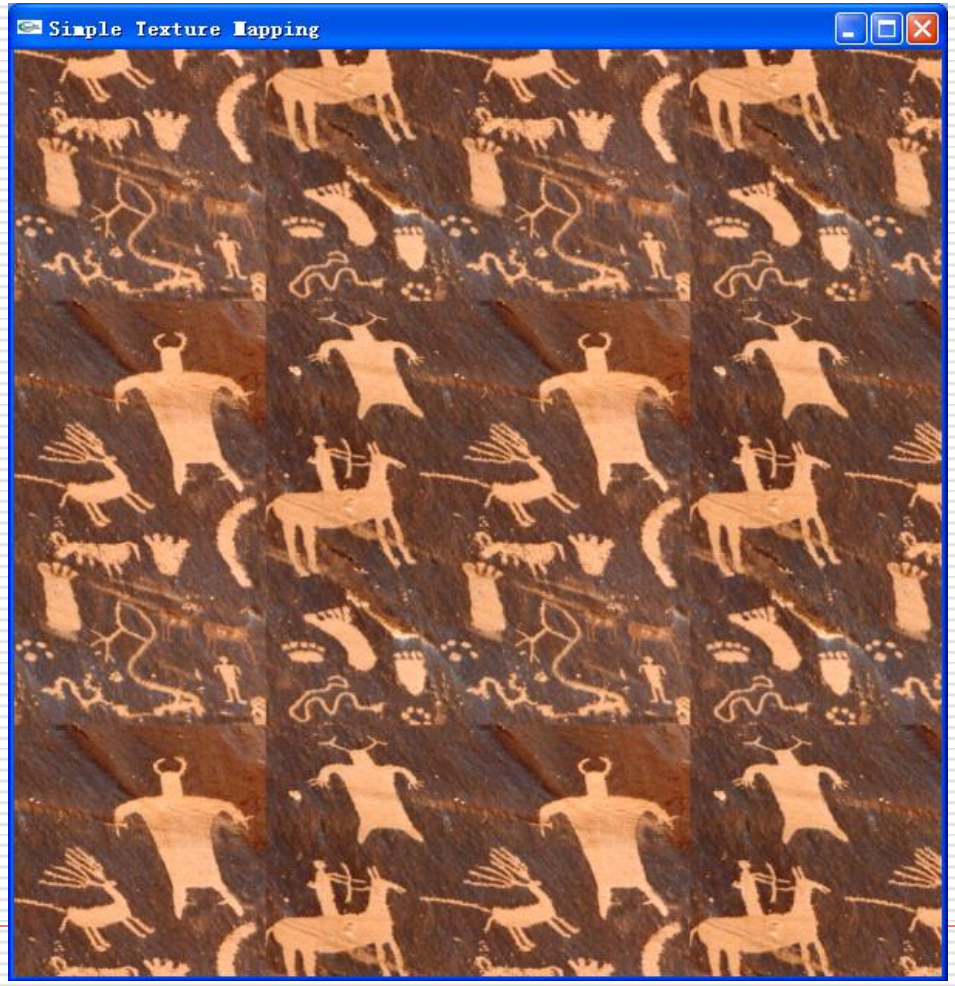
GL_CLAMP wrapping

Step 3. Texture functions



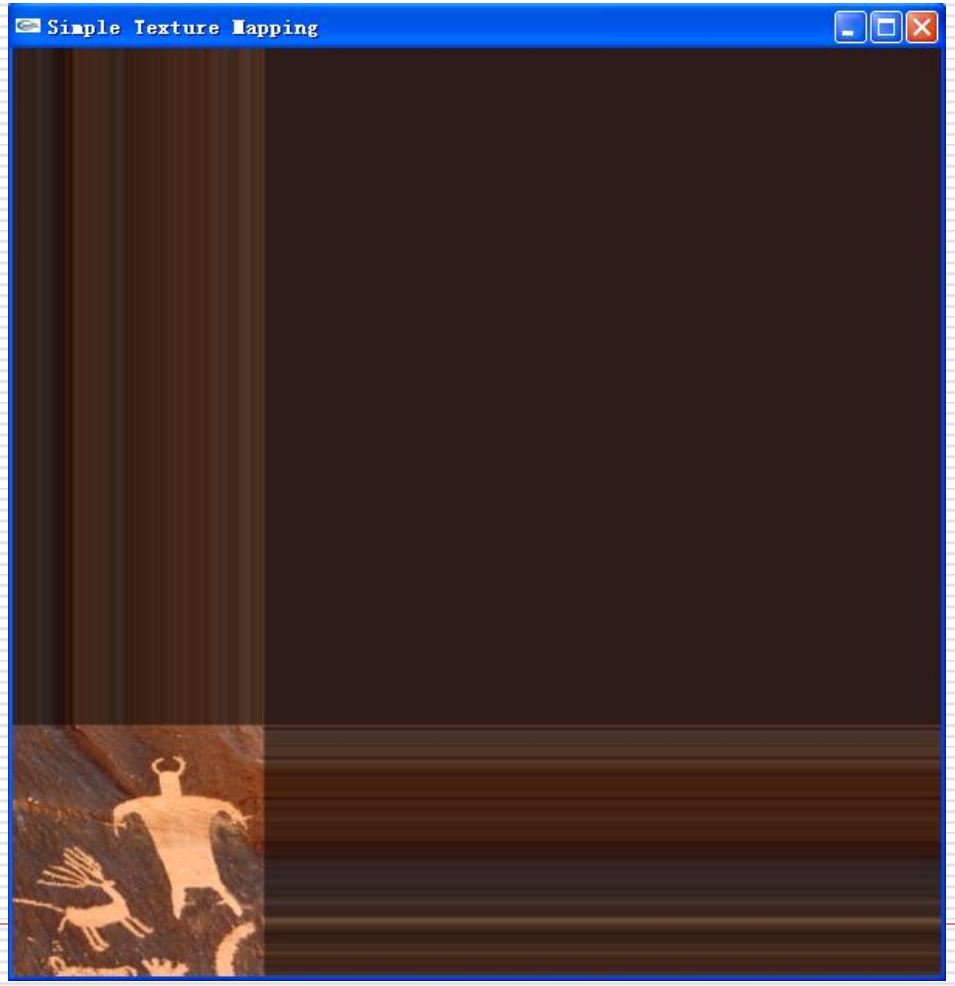
Step 3. Texture functions

GL_REPEAT



Step 3. Texture functions

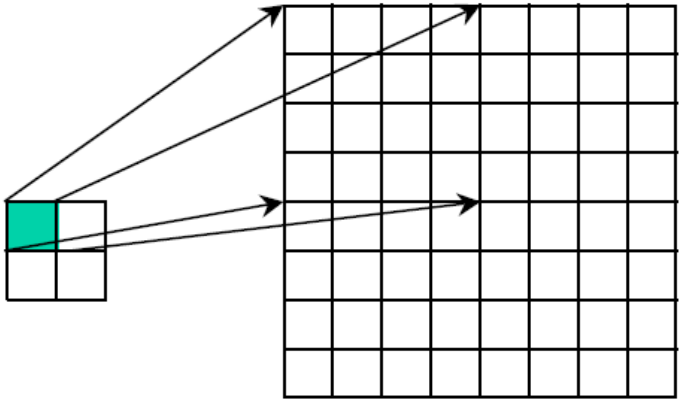
GL_CLAMP



Magnification and minification

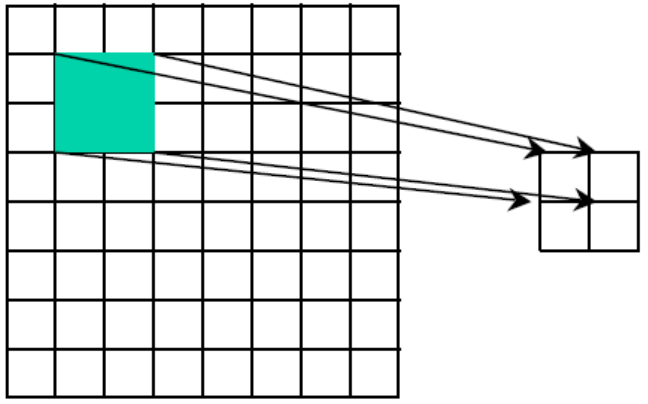
More than one texel can cover a pixel (*minification*) or more than one pixel can cover a texel (*magnification*)

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



Texture Polygon

Magnification



Texture Polygon

Minification

Filter mode

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)

Mipmapped textures

- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Lessens interpolation errors for smaller textured objects
- Declare mipmap level during texture definition

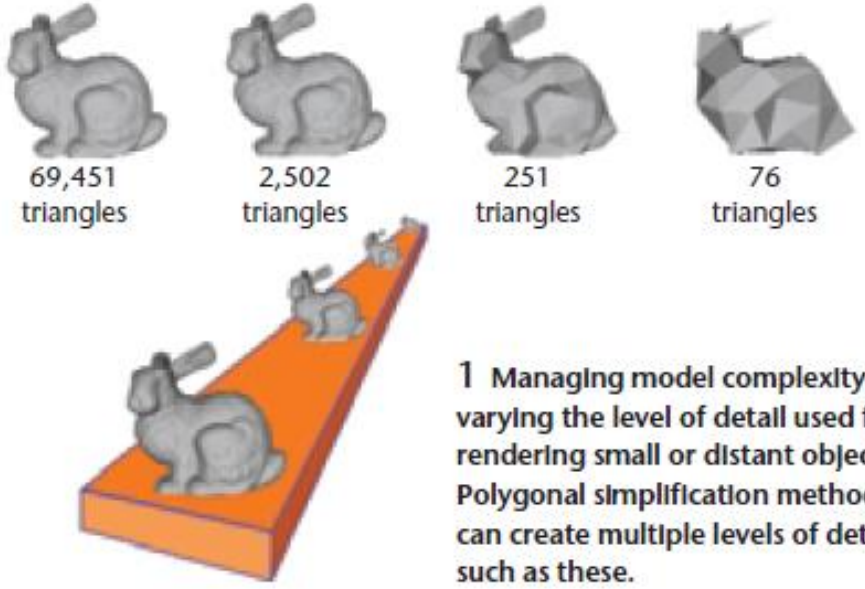
```
glTexImage2D( GL_TEXTURE_*D, level, … )
```

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

```
gluBuild*DMipmaps( … )
```

Geometry: Level-of-details

- Level-of-Details (LOD)
  When the object is far
  from the viewpiont,
  using the low-resolution;
  When the object is near,
  using the fine resolution

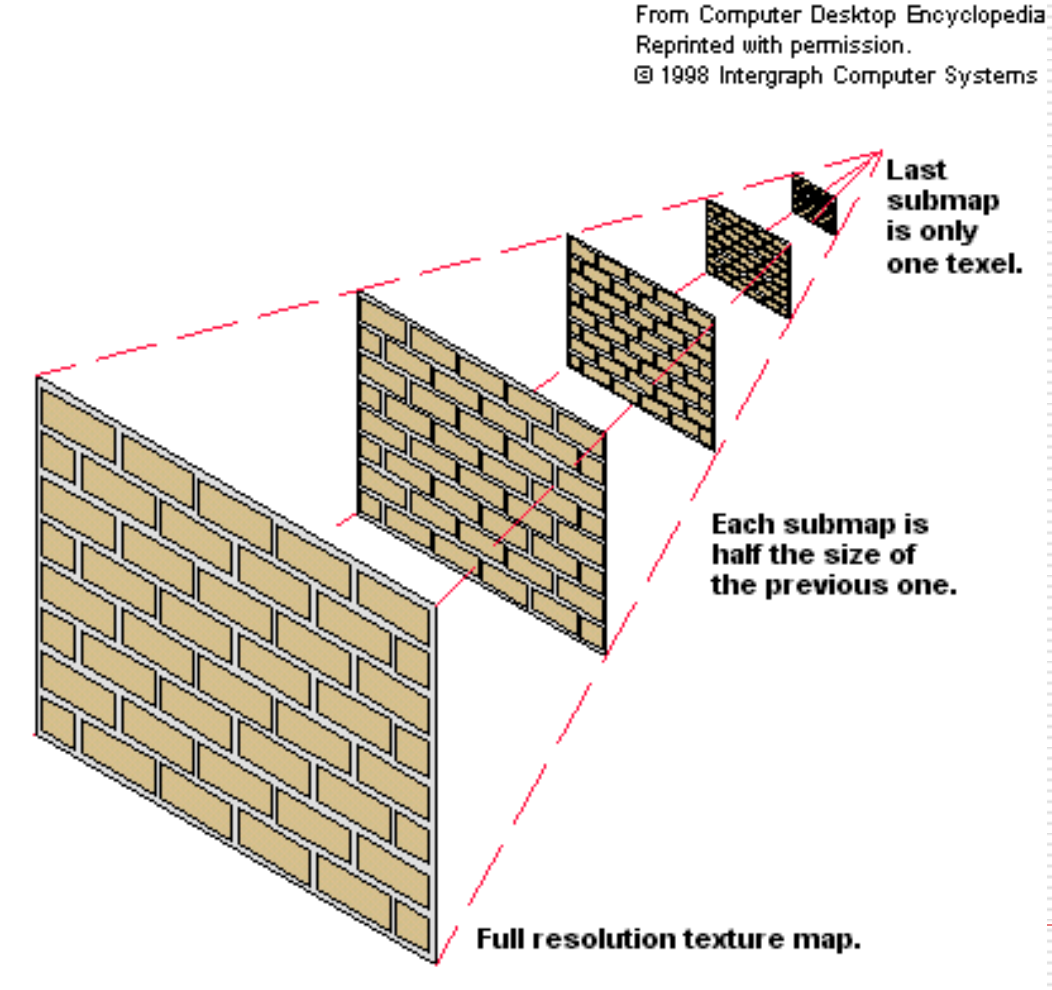


1 Managing model complexity by varying the level of detail used for rendering small or distant objects. Polygonal simplification methods can create multiple levels of detail such as these.

Texture: Level-of-details

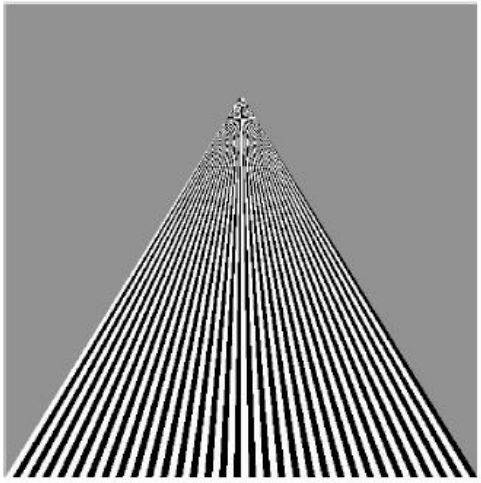
In 3D computer graphics texture filtering, mipmaps (also MIP maps) are pre-calculated, optimized collections of images that accompany a main texture, intended to increase rendering speed and reduce aliasing artifacts. They are widely used in 3D computer games, flight simulators and other 3D imaging systems. The technique is known as mipmapping. The letters "MIP" in the name are an acronym of the Latin phrase multum in parvo, meaning "much in little". Mipmaps need more space in memory. They also form the basis of wavelet compression

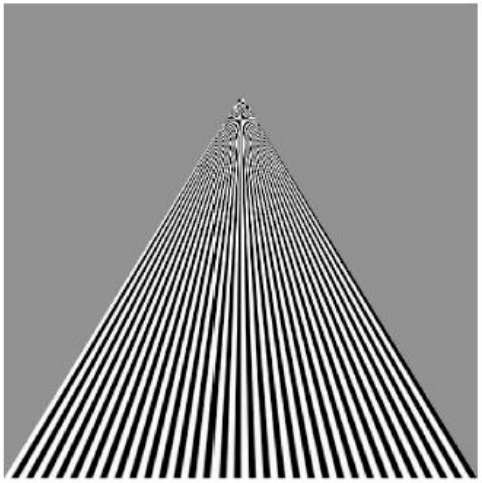
Texture: Level-of-details



Example

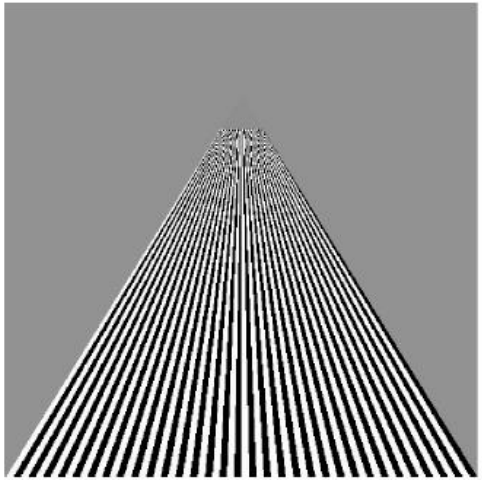
point sampling

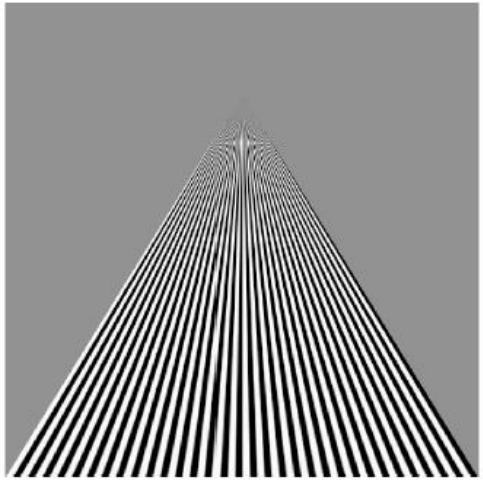




linear filtering

mipmapped point sampling



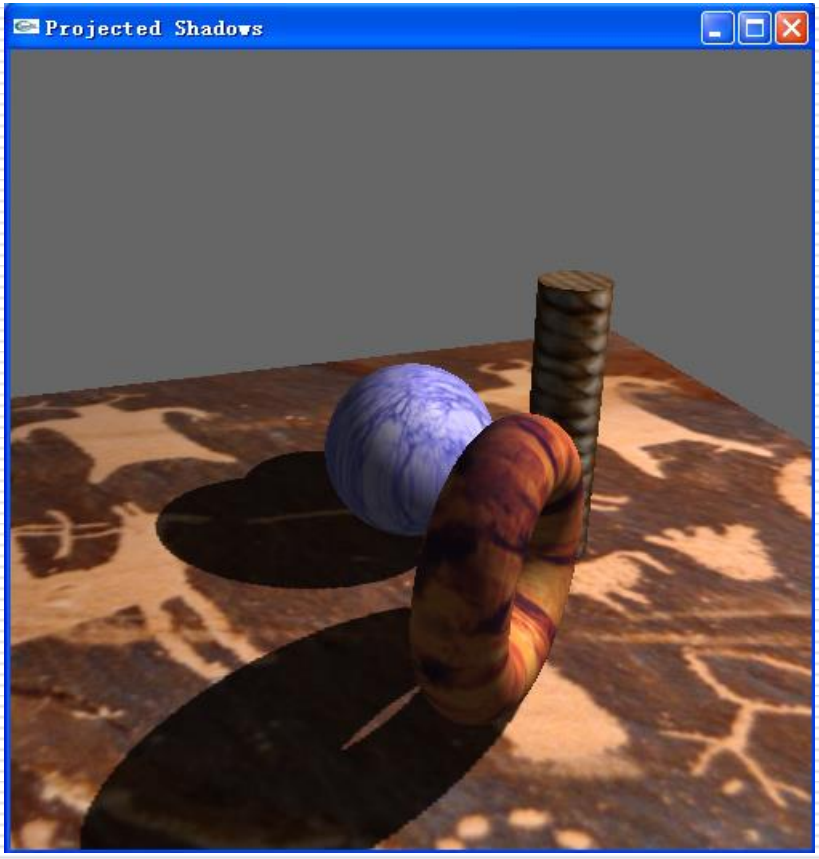


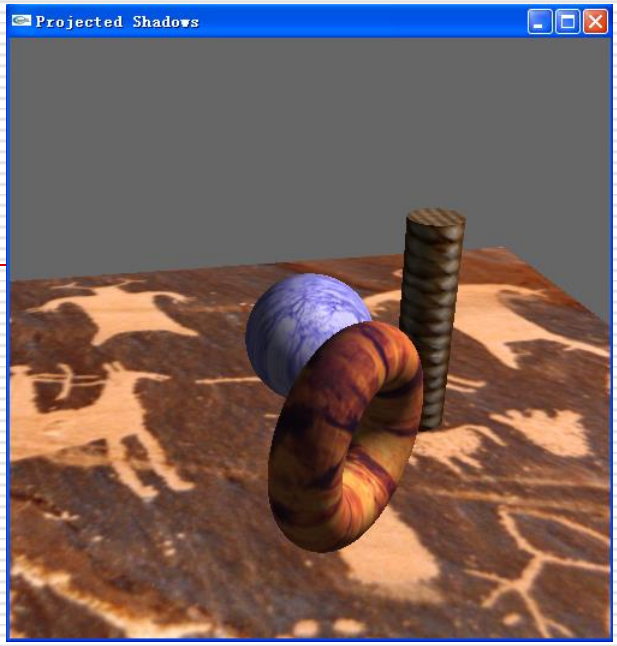
mipmapped linear filtering

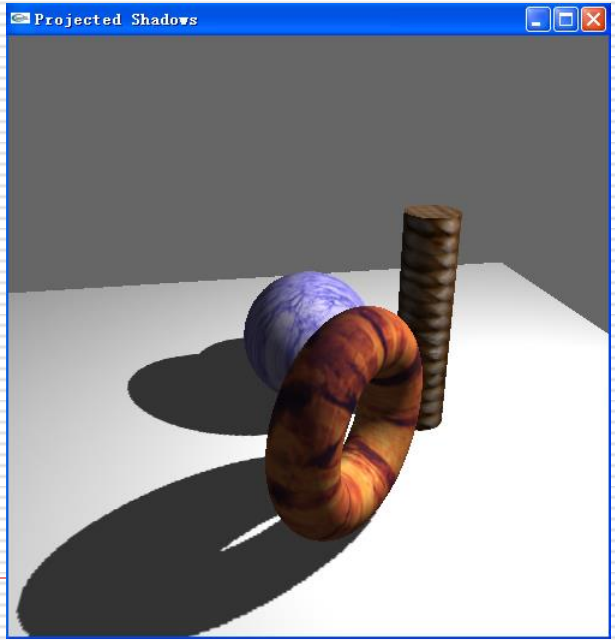
Step 3. Texture functions

- 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`

Texture blending







Perspective correction hint

- Texture coordinate and color interpolation
  - either linearly in screen space
  - or using depth/perspective values (slower)
- Noticeable for polygons “on edge”
  - **`glHint( GL_PERSPECTIVE_CORRECTION_HINT, hint )`**

    where **`hint`** is one of
    - **`GL_DONT_CARE`**
    - **`GL_NICEST`**
    - **`GL_FASTEST`**

Generating texture coordinates

- 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)

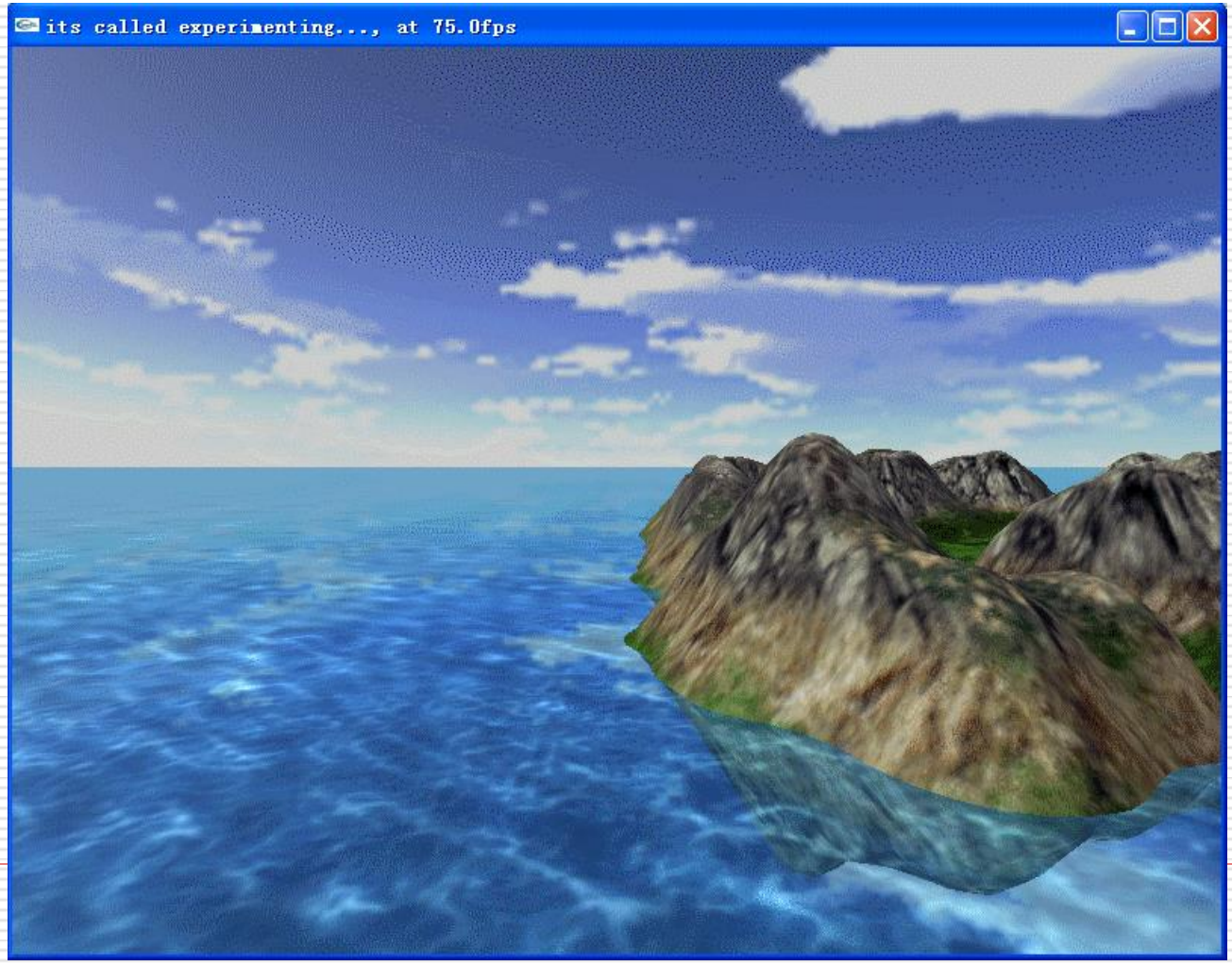
Texture objects

- 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

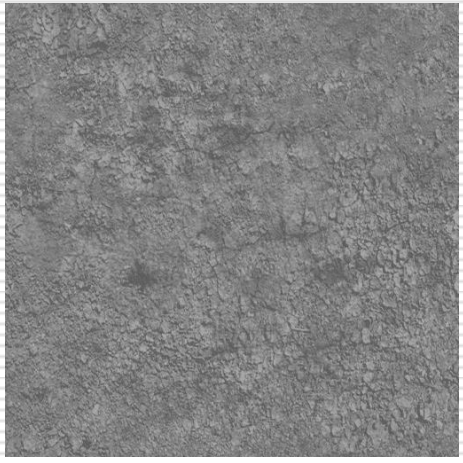
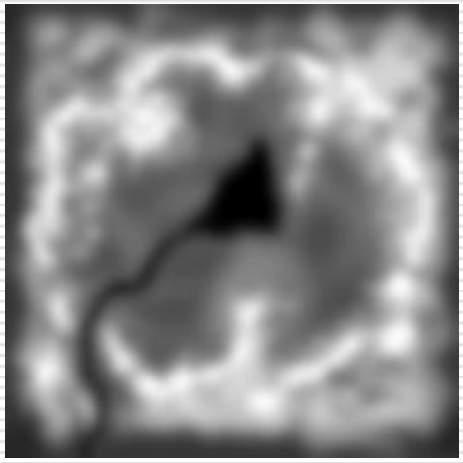
Outline

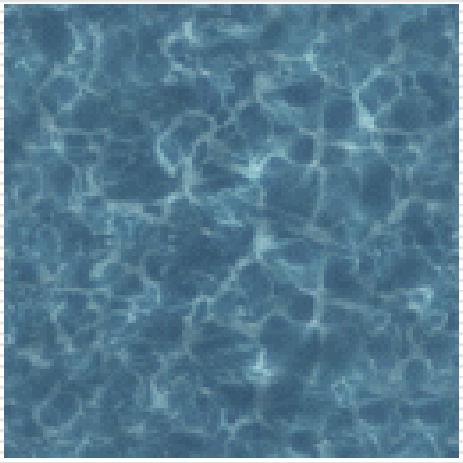
- What is texture mapping?
- OpenGL texture functions
- Texture mapping applications

Course project



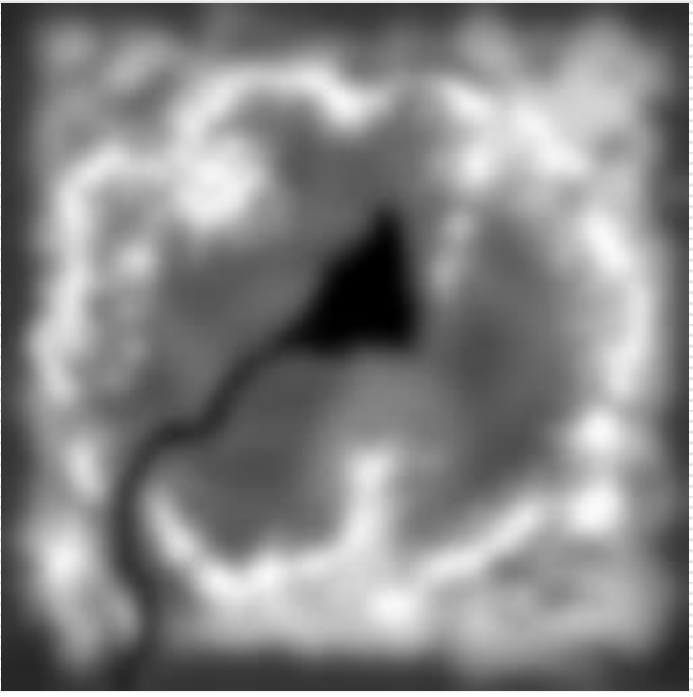
Course project

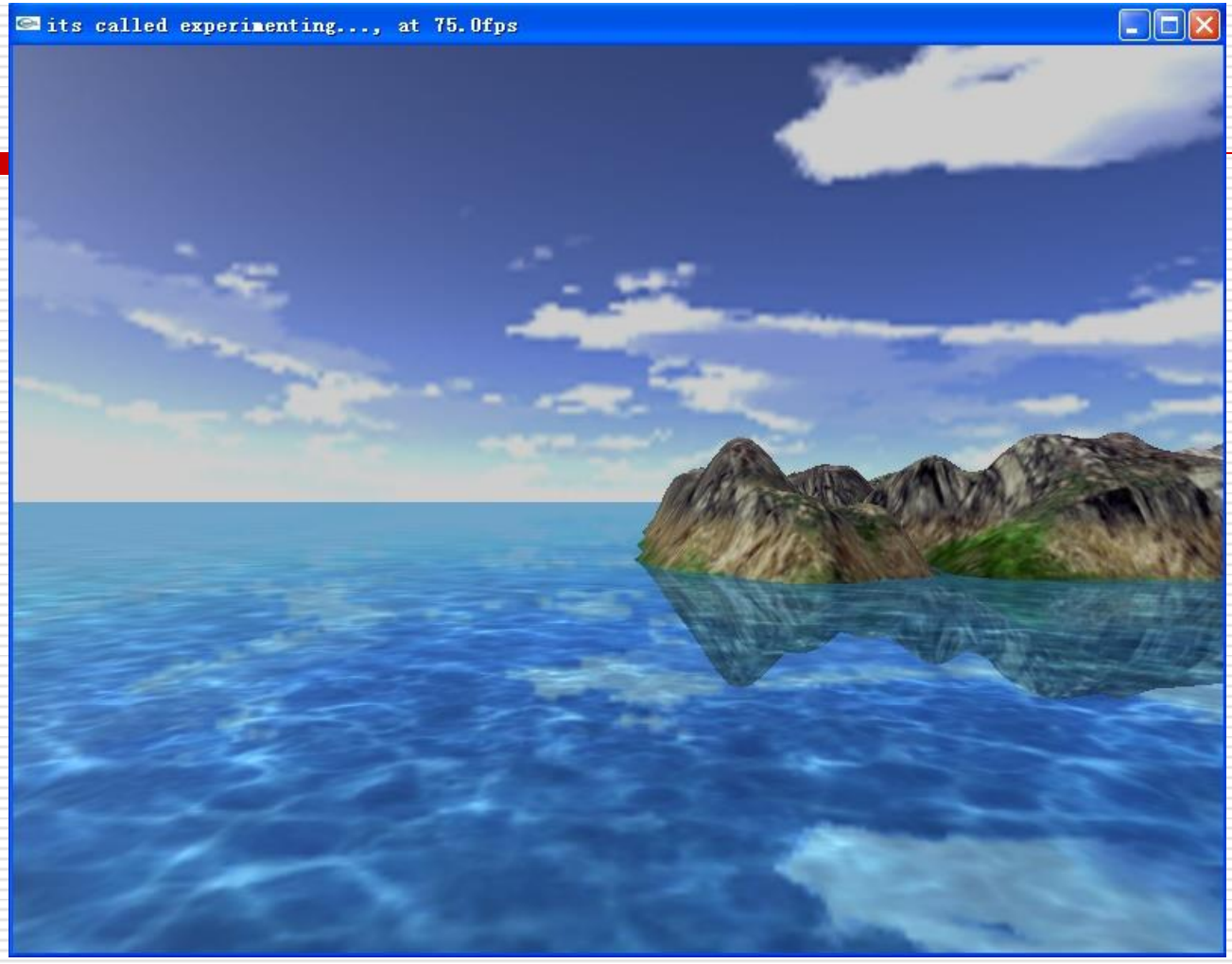





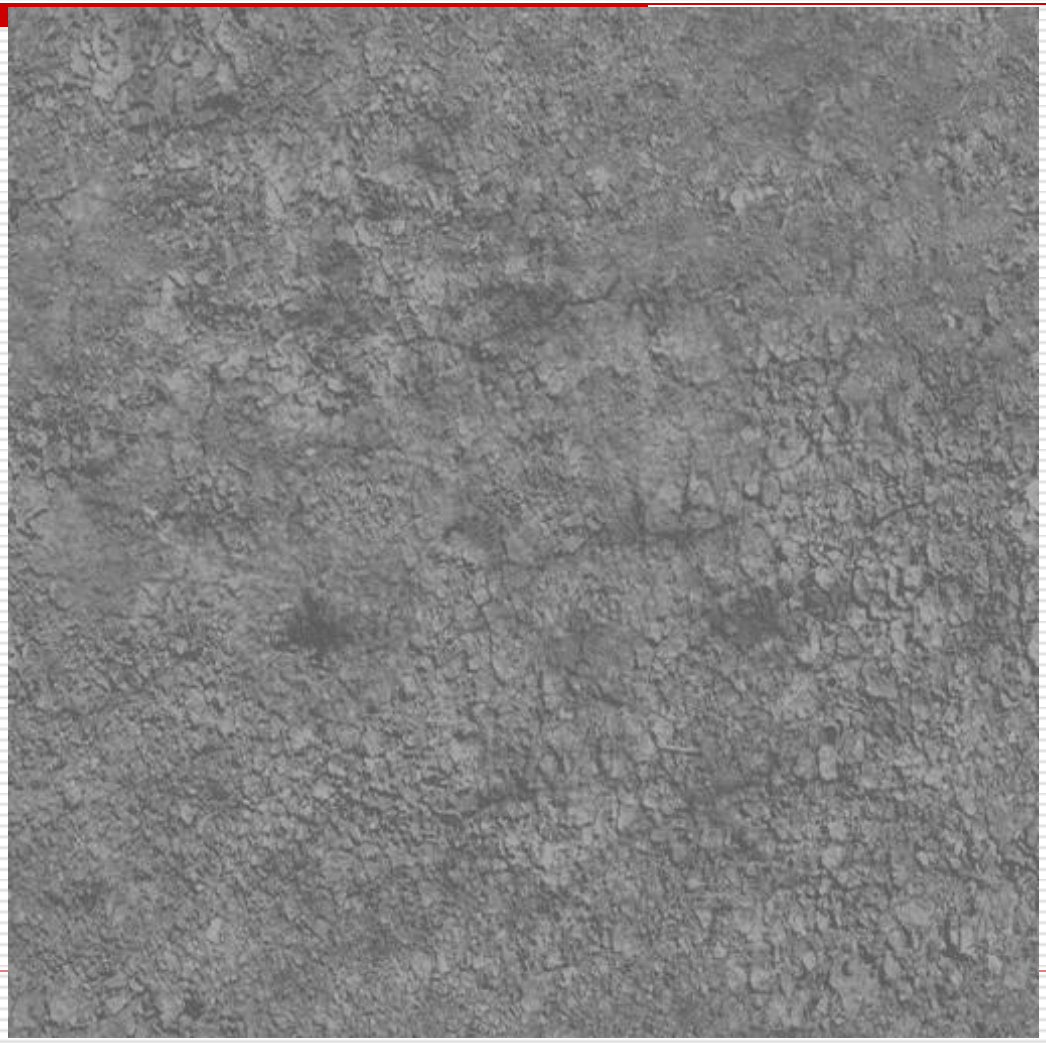







its called experimenting..., at 75.0fps

Course project

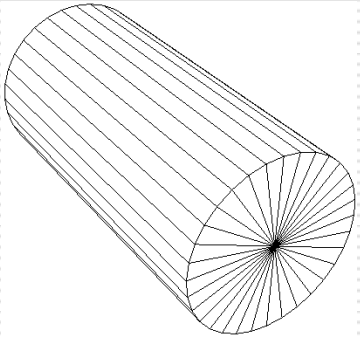


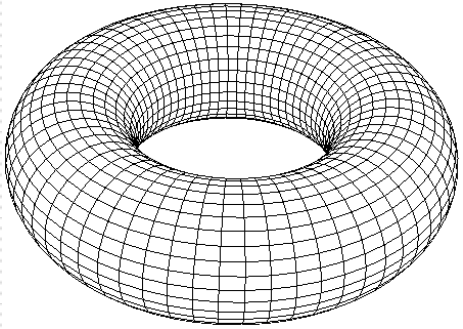

its called experimenting..., at 75.0fps

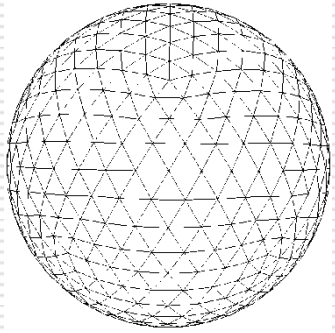

its called experimenting..., at 69.1fps

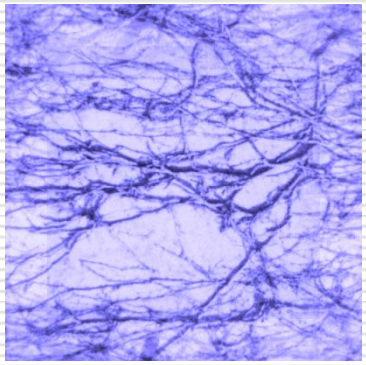
Course project

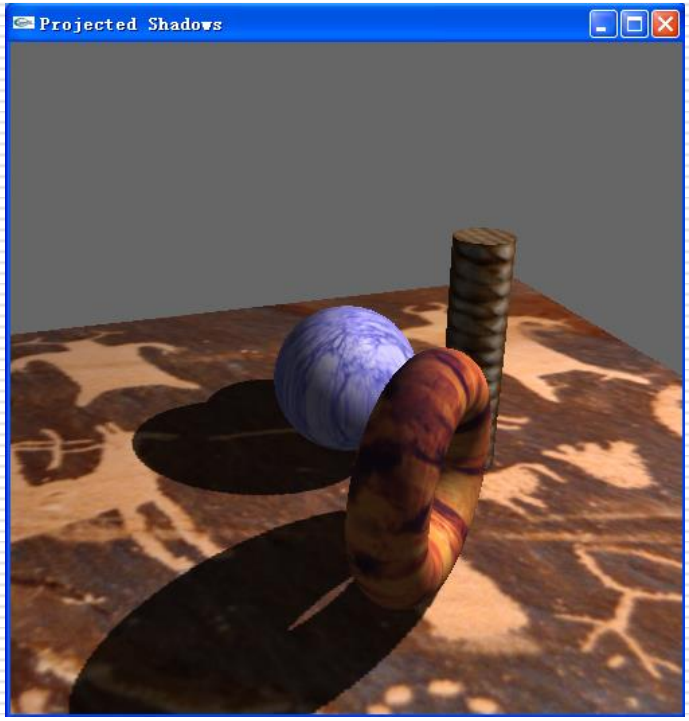
Using parametric domain?



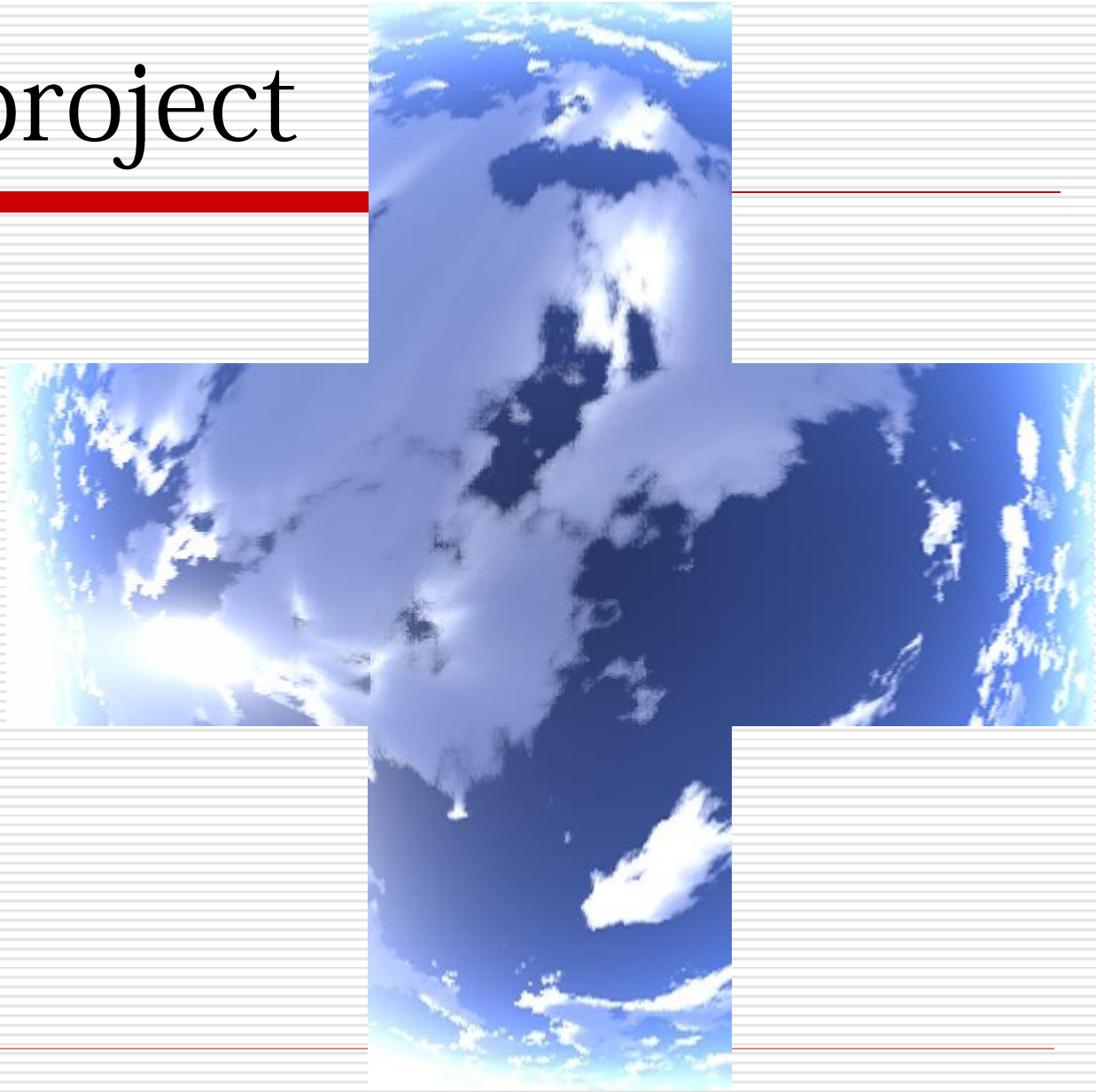




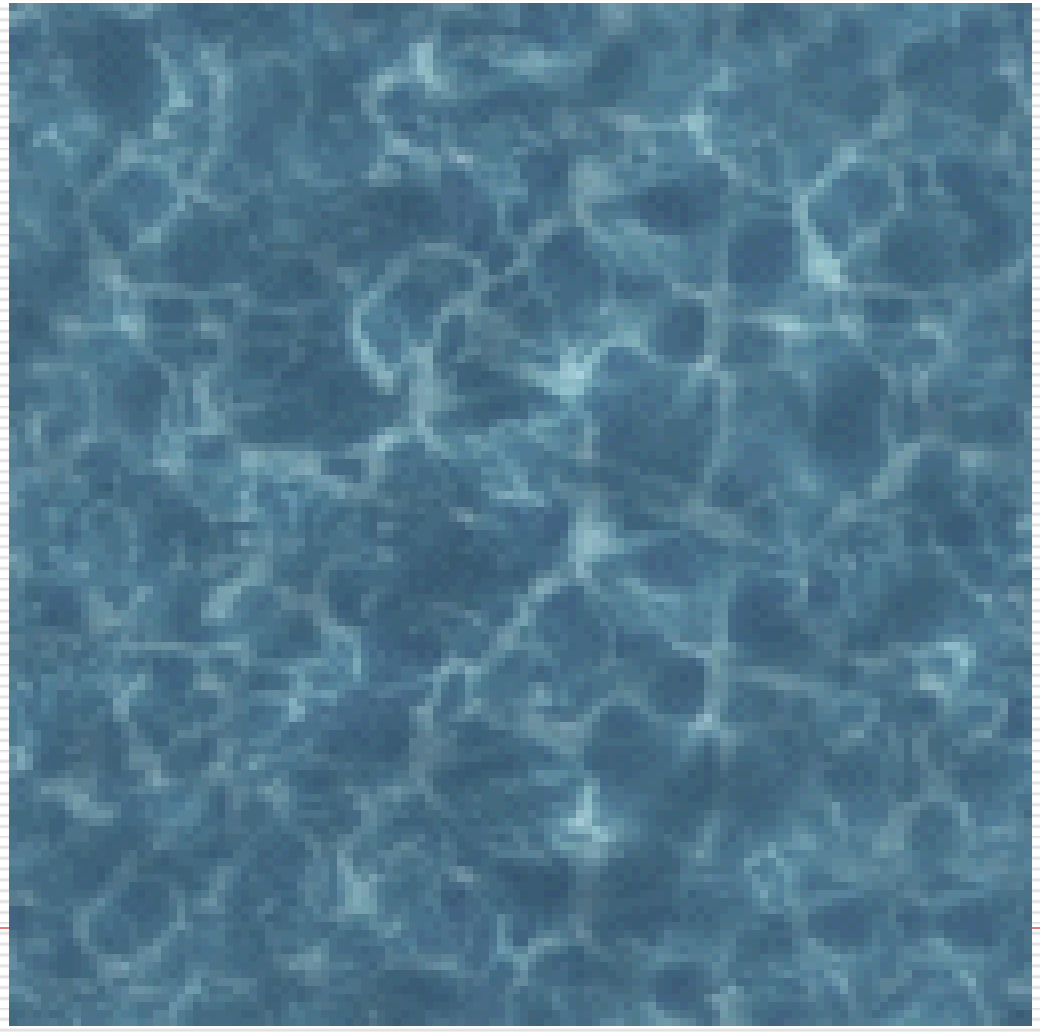




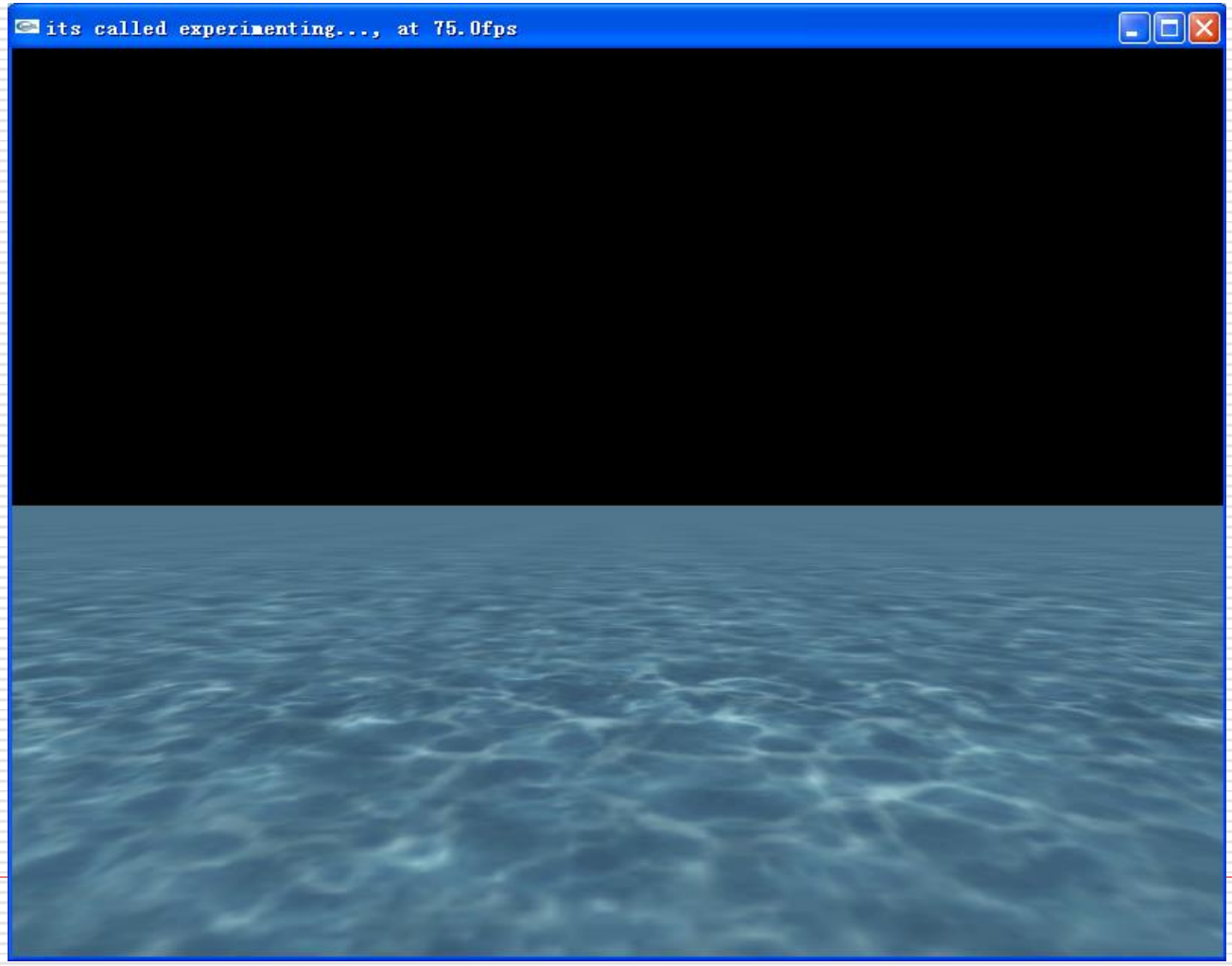
Course project



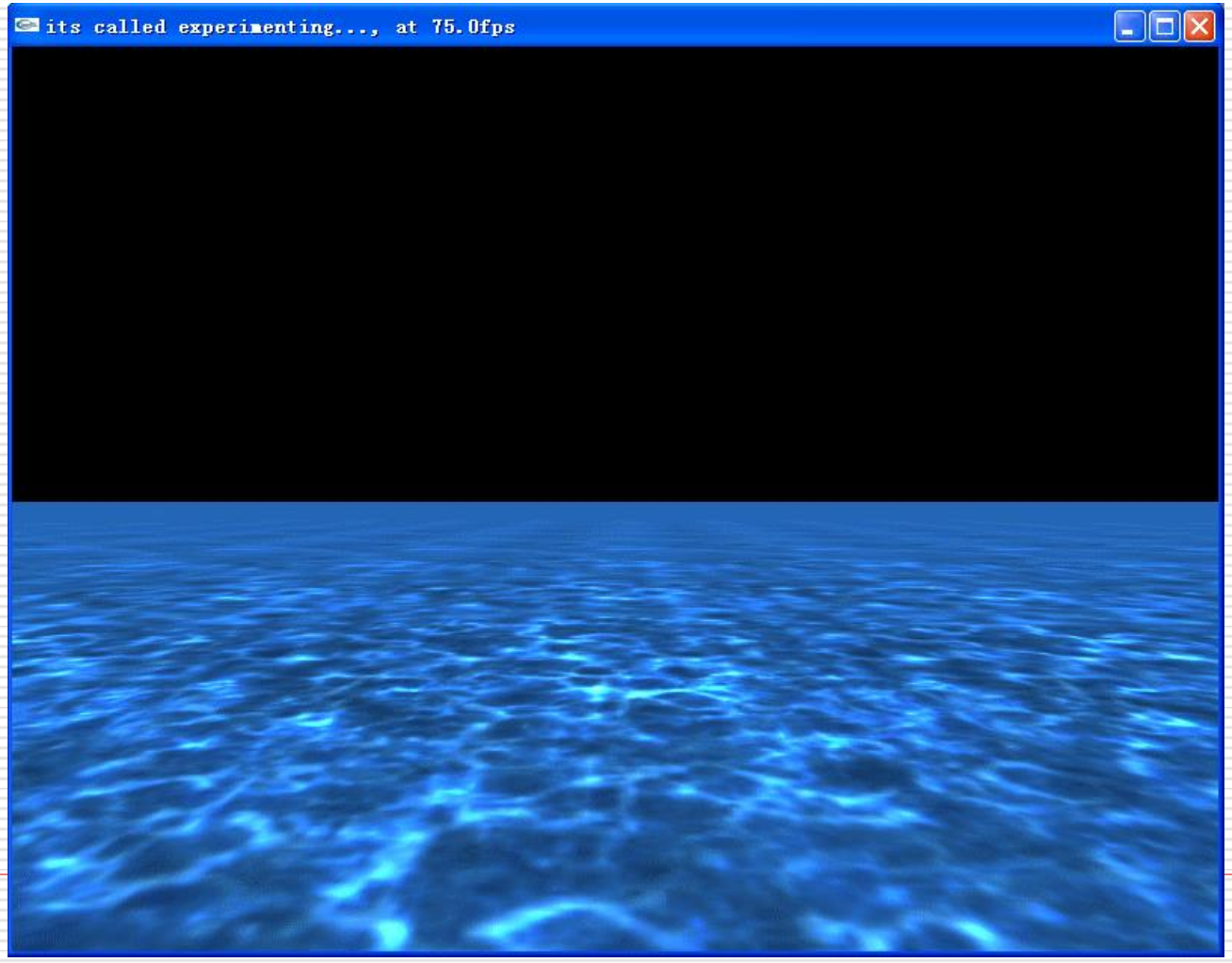
Course project



Course project



Course project



Initial coordinate (x, y)

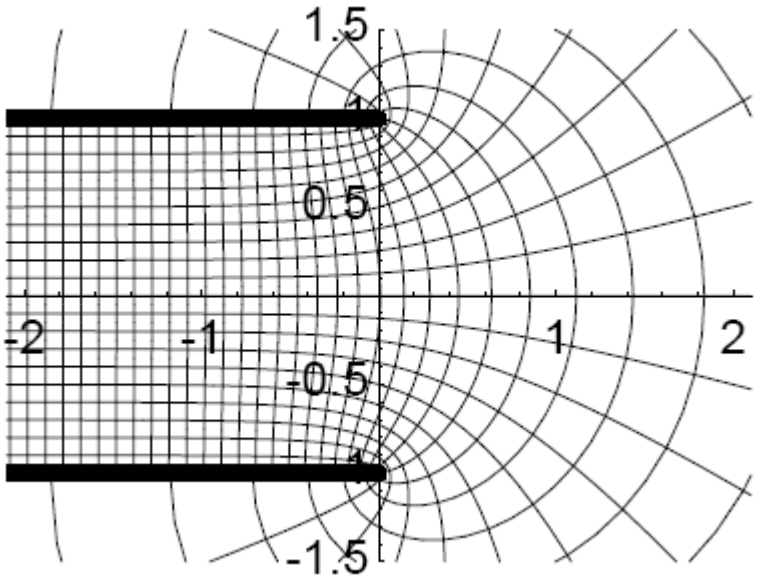
$$z = x + iy$$

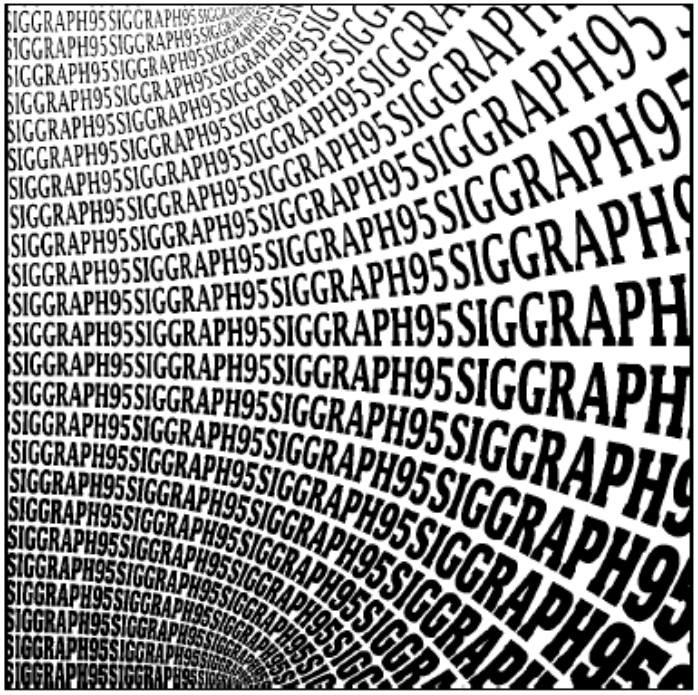
Coordinate after transformation (u, v)

$$w = u + iv$$

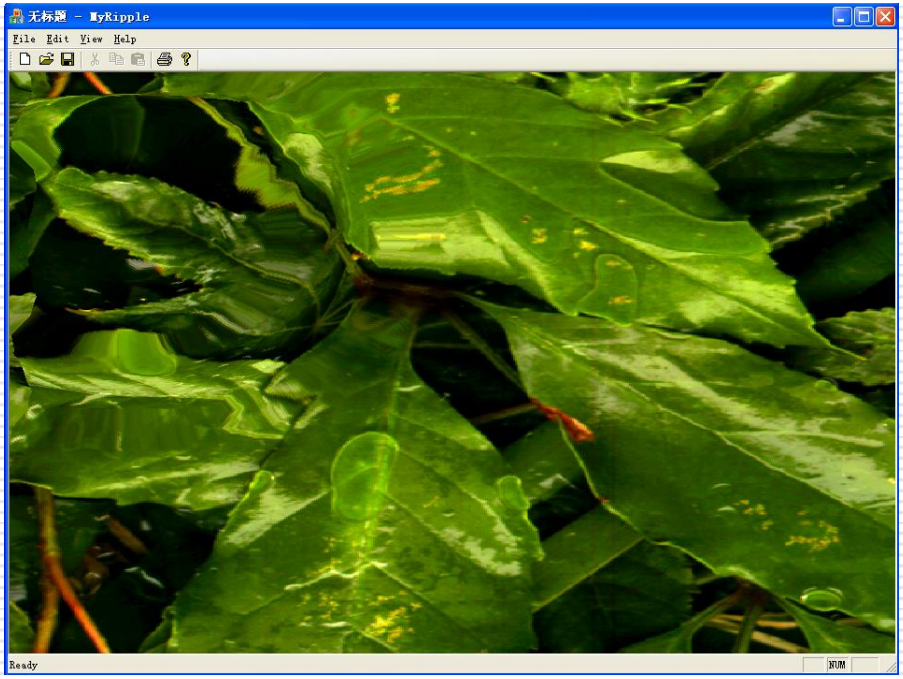
Conformal mapping: Angle preserving transformation

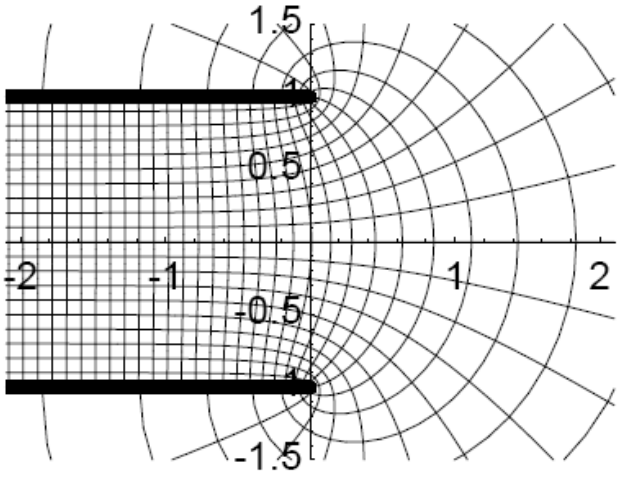
$$w = k(1 + z + e^{z})$$

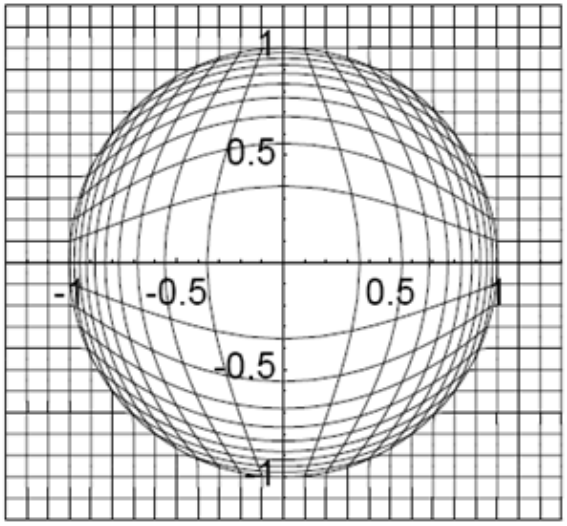




Transform who?
texture or geometry

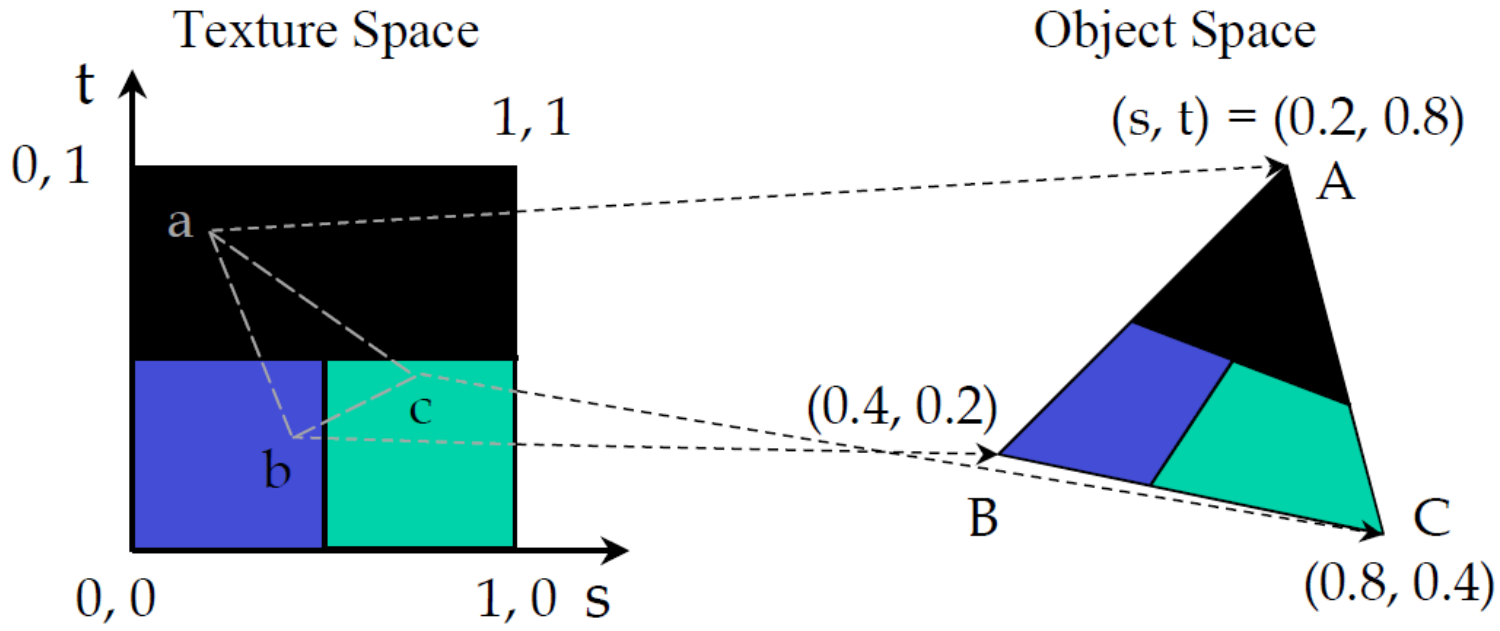




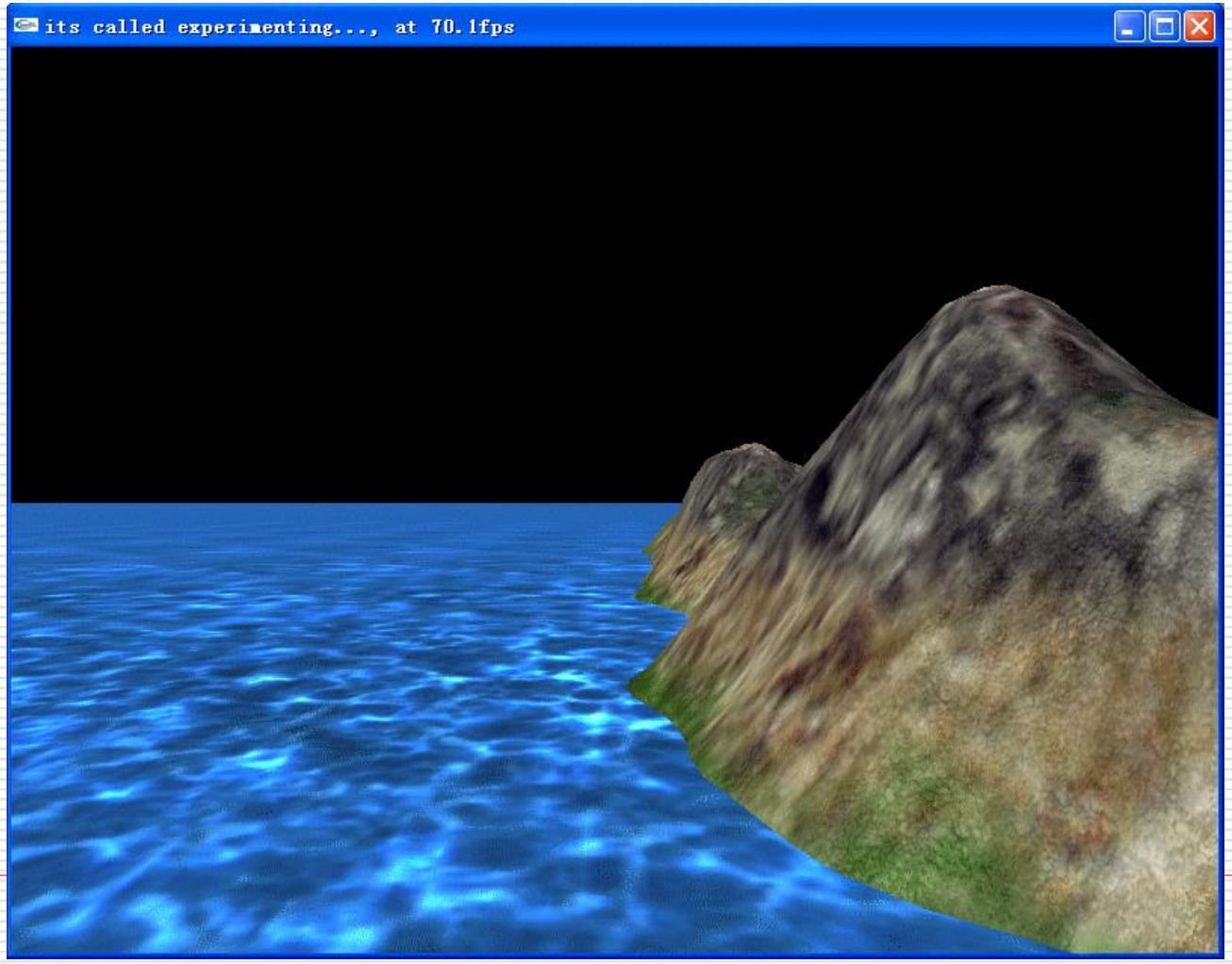


Step 2: Mapping a texture

- Based on parametric texture coordinates
- `glTexCoord*()` specified at each vertex

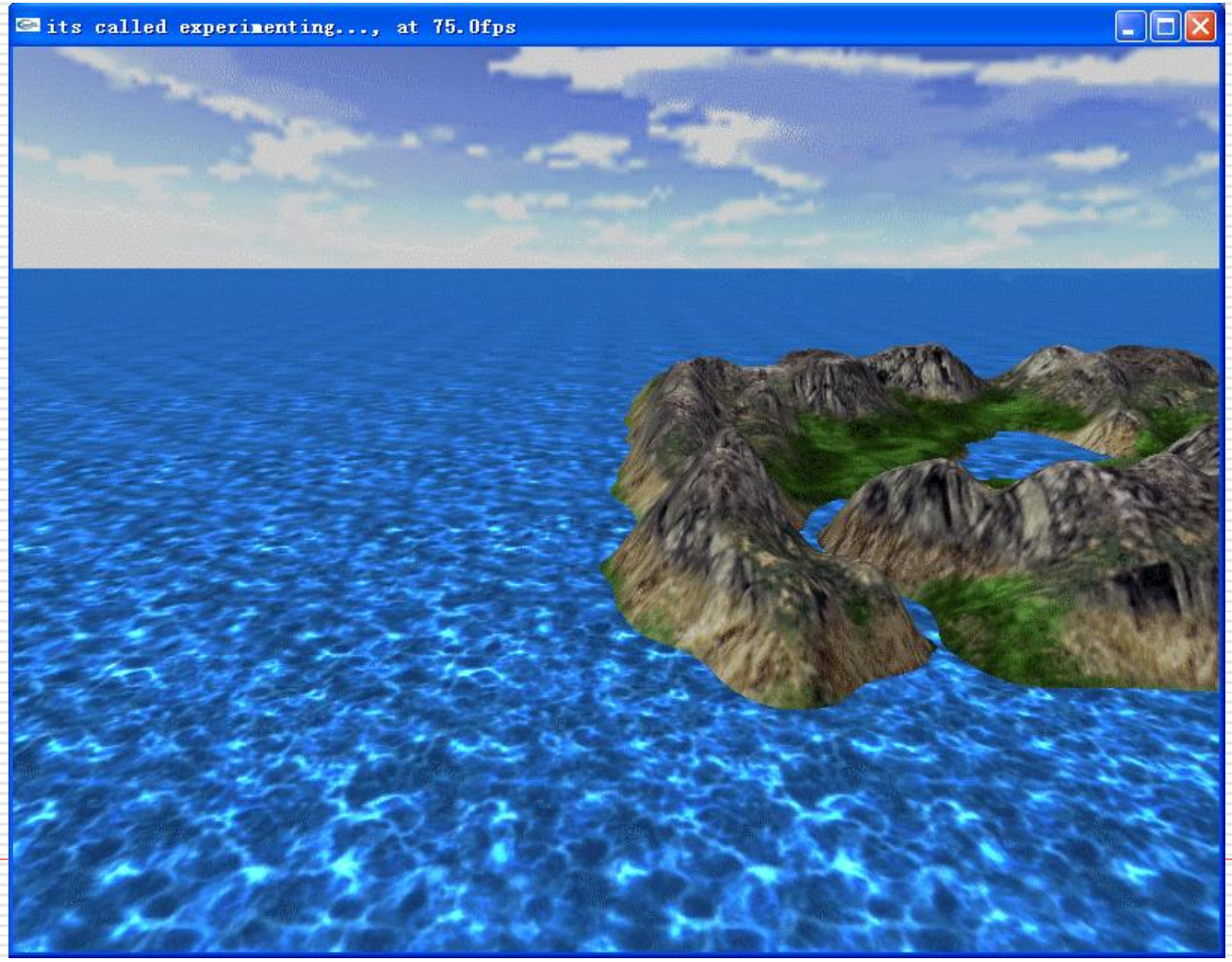


Course project

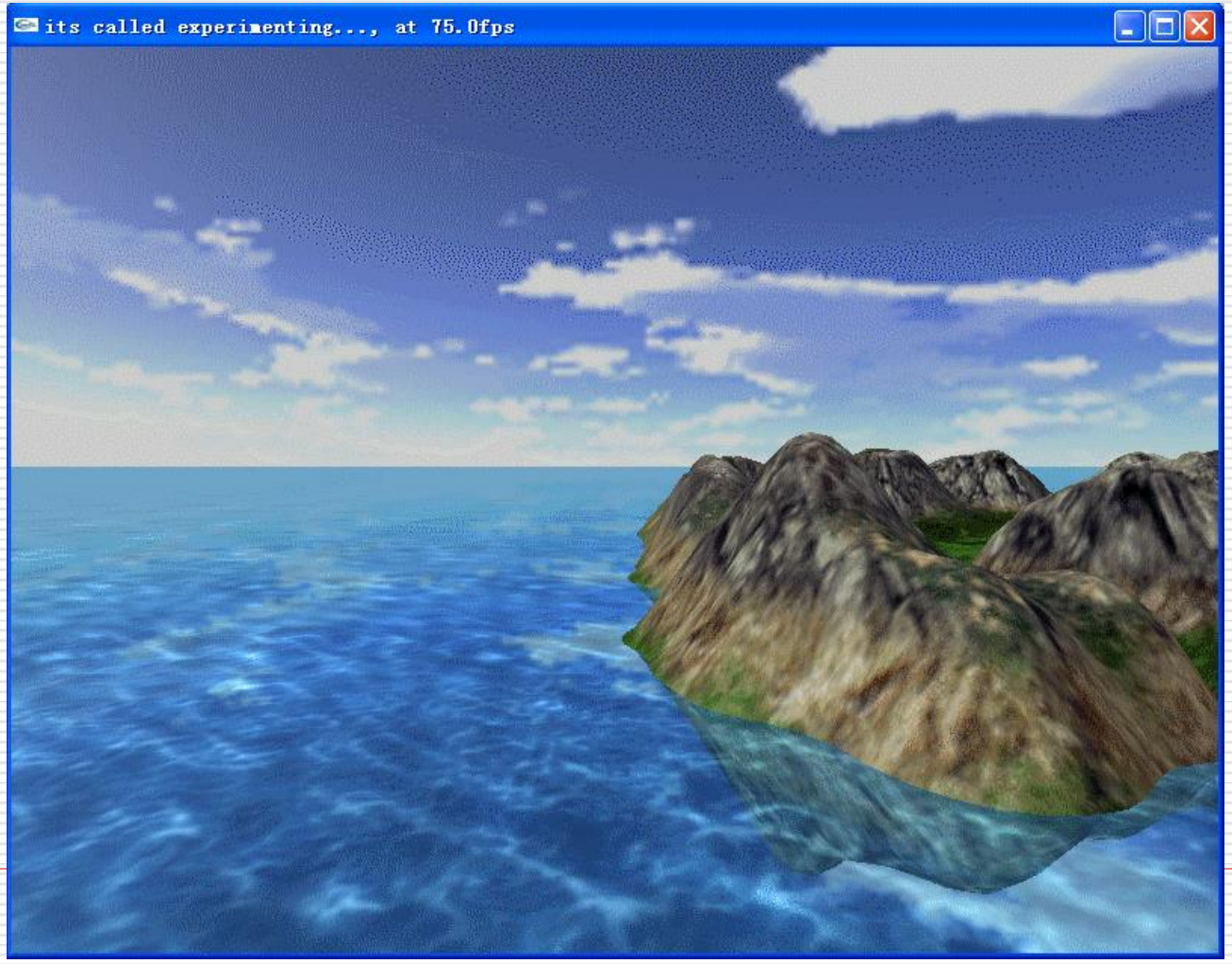


Course project

Course project

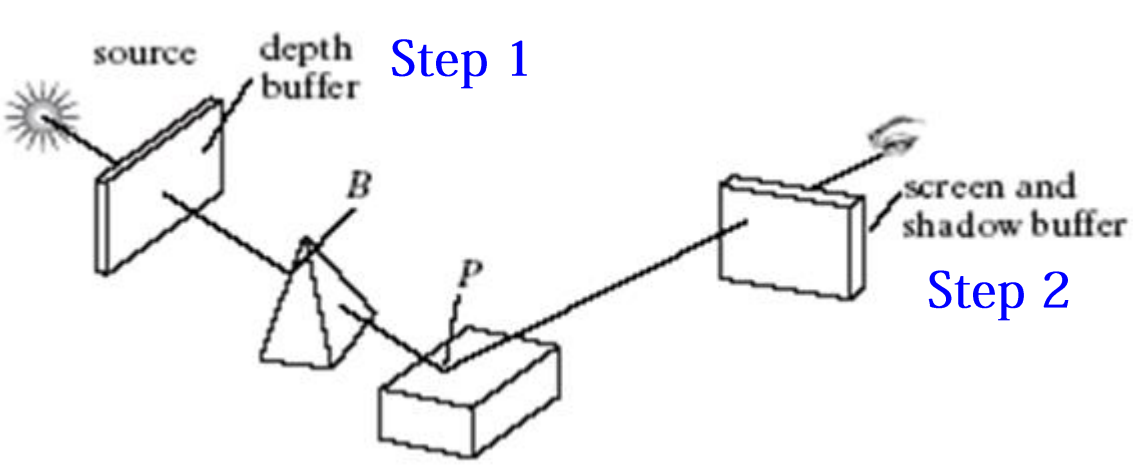


Course project



(2) Shadows Using a Shadow Buffer

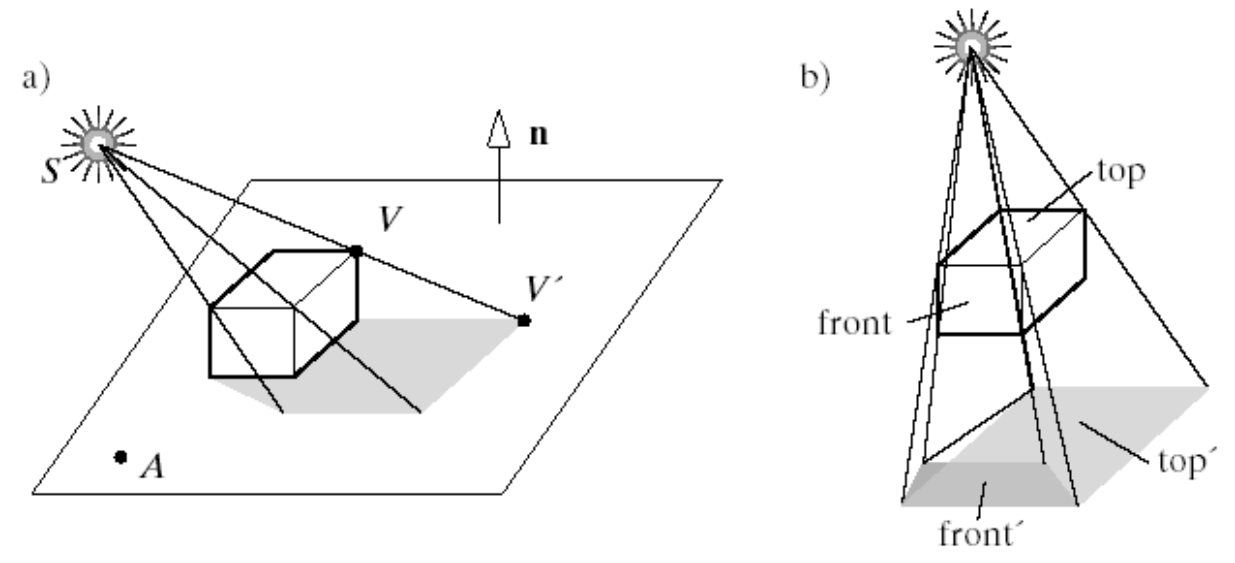
- This method works for non-planar surfaces
- The general idea is that any point hidden from the light source is in shadow
- A second depth buffer (the shadow buffer) is used. It contains depth information about the scene from the point of view of the light source



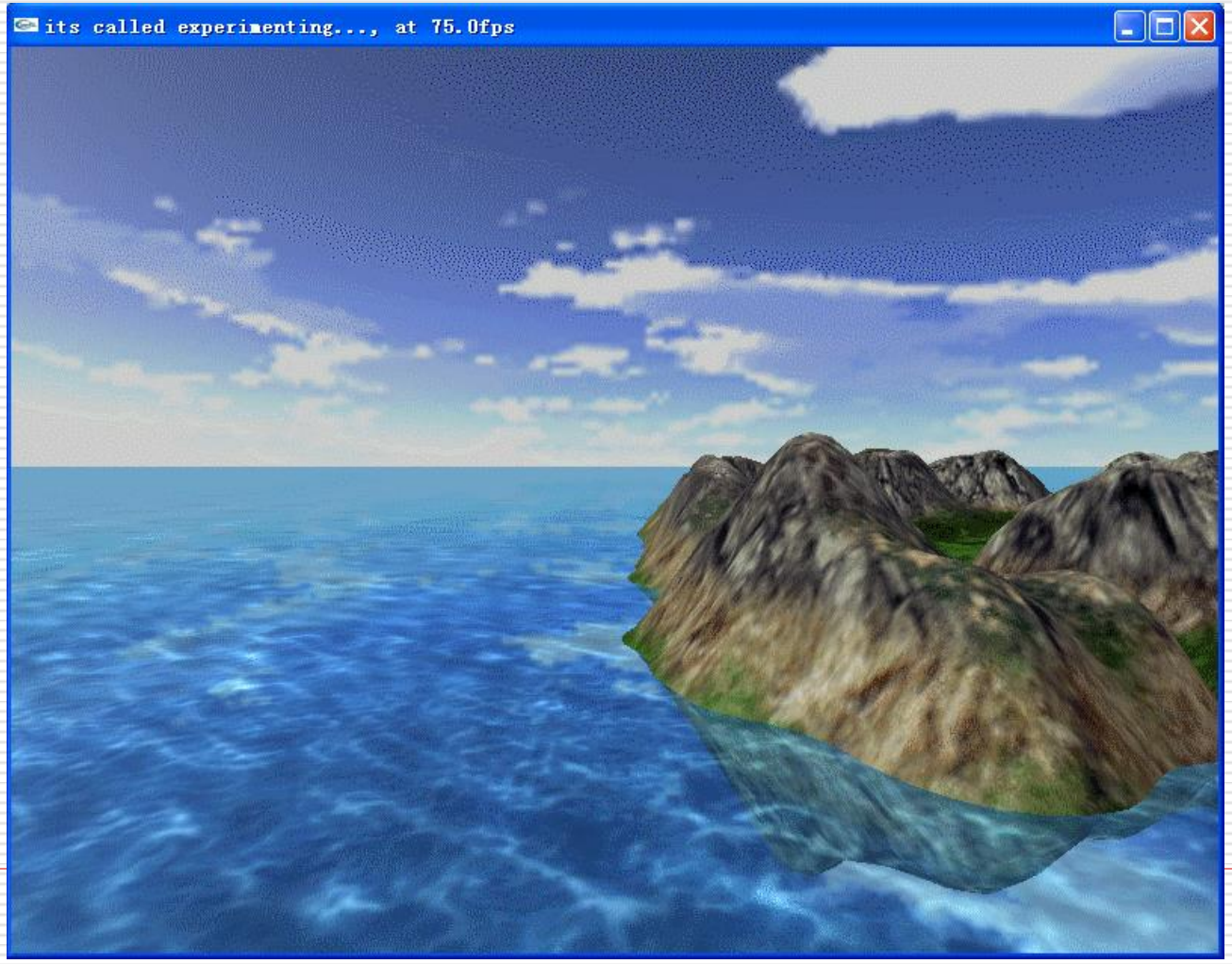
(1) Shadows as Texture

The problem is to compute the shape of the shadow that is cast

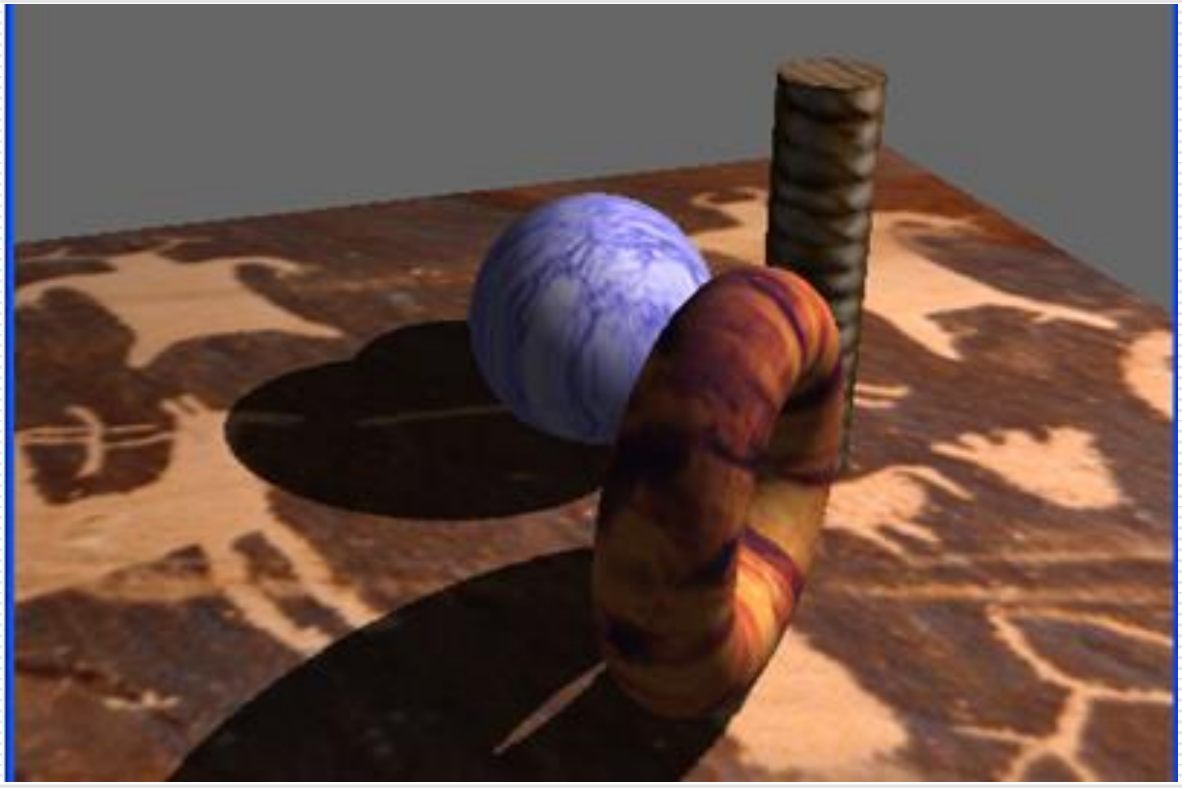
- The shape of the shadow is determined by the projections of each of the faces of the box onto the plane of the floor using the source as the center of projection. The shadow is the union of the projections of the six faces



Course project



Can we do this?



End.
Thanks