Computer Graphics

Texture Mapping

Teacher: A.prof. Chengying Gao(高成英)

E-mail: mcsgcy@mail.sysu.edu.cn

School of Data and Computer Science

Texture

- Texture mapping(texturing)is a fundamental technique in Computer Graphics, allowing us to represent surface details without modeling geometric material details.
- We will describe the basic ideas and applications of texture in this course

Limitations

- So far, every object has been drawn either in a solid color, or smoothly shaded between the colors at its vertices.
  - Similar to painting

Generated with Blue Moon Rendering Tools — www.bmrt.org

Why we need texture?

- Modeling surface details is one of the most important tasks for rendering realistic images.
  - For example, if we want to model a brick wall.
  - One option: use a huge number of polygons with appropriate surface coloring and reflectance characteristics to model the surface details

Limitations

- Even though the graphics card can display up to ten million polygons per second,It's difficult to simulate all the phenomenon in nature.
  - Cloud
  - Grass
  - Landforms
  - Skin
  - Leaf
  - Hair
  - Fire and Water
  - ……

Limitations

- Representing all detail in an image with polygons would be cumbersome

Specific details

Structured noise

Pattern w/ randomness

Section through volume

Bumps

3

Why we need texture?

- However, in most applications, few people would care much about the details of the brick wall and would normally view the wall from a distance.
- In this situation, we probably don’t need to know all the details of the brick wall. Instead, we can simply model the wall as a big flat polygon, and paste onto the polygon a wall image so that the polygon looks like a real brick wall.

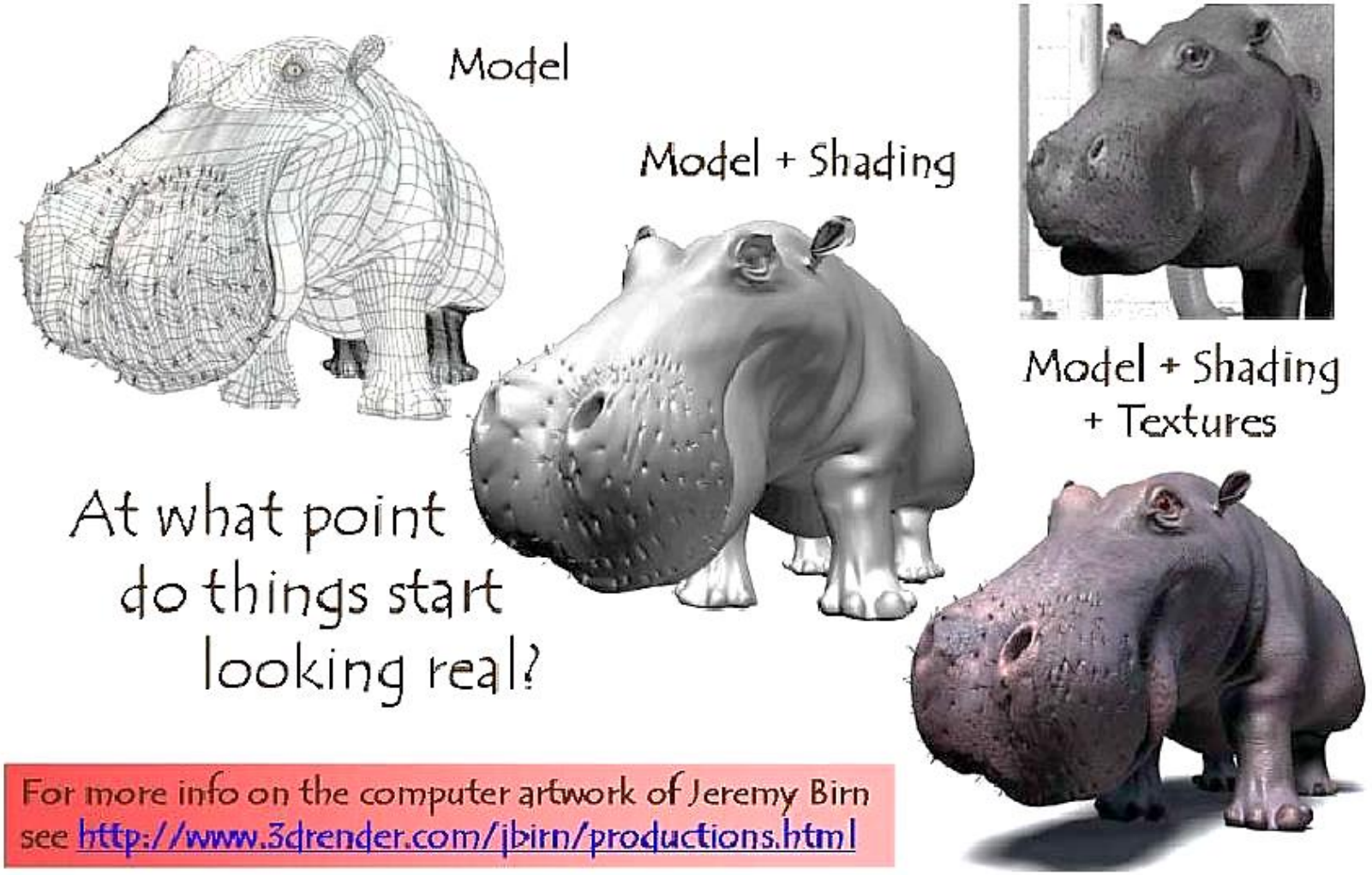
What is texture?

- This image, which is an approximation to the real brick wall, is called texture in graphics
- The process of applying the texture to an object surface is called texture mapping (纹理映射) or texturing (贴纹理).

Why Texture is Importance ？

- Texturing resolves the two problems：
  - First, by representing the surface as a texture image, you don’t have to painfully model all the geometric and material details. This saves time and resources and allow users to do other more important things.
  - Second, by rendering a rough polygonal model (e.g. a single square polygon for a brick wall) and a texture instead of a detailed geometrical model with different BRDFs, the rendering can be done much more efficiently. This saves computers time and resources.

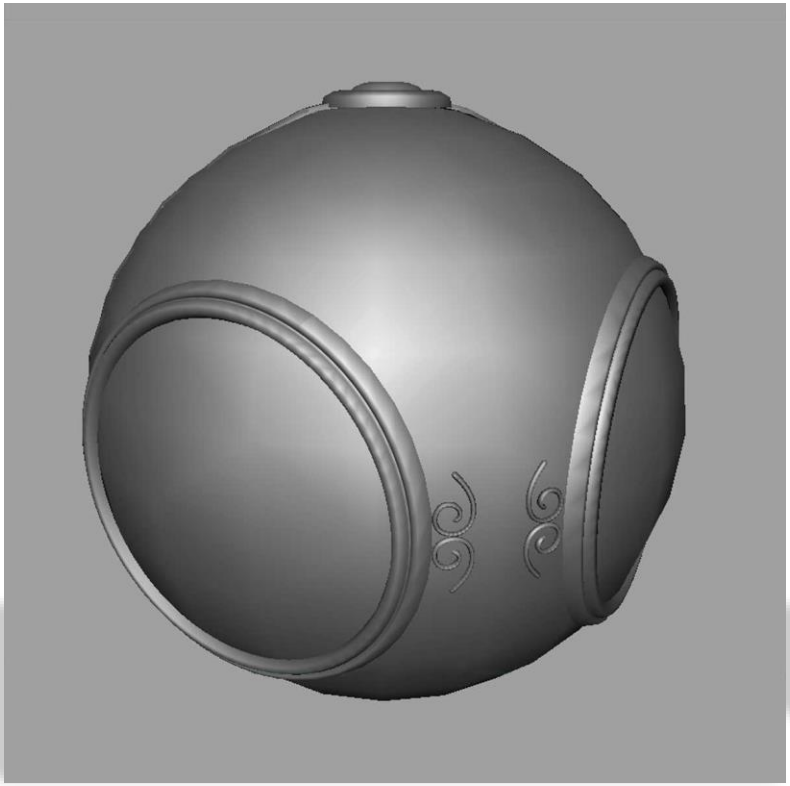
The Quest for Visual Realism

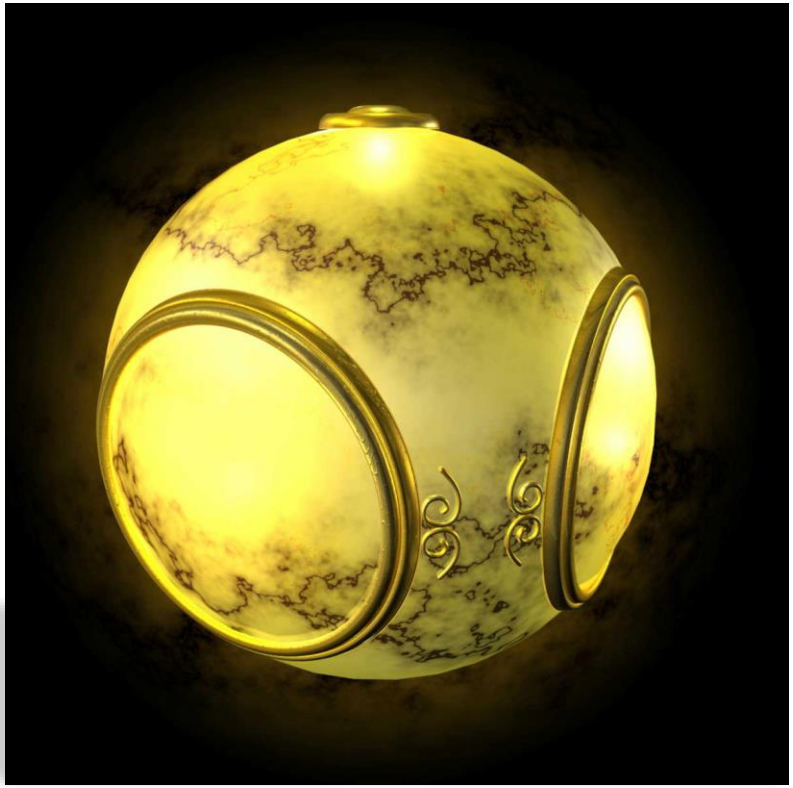


Three Types of Mapping

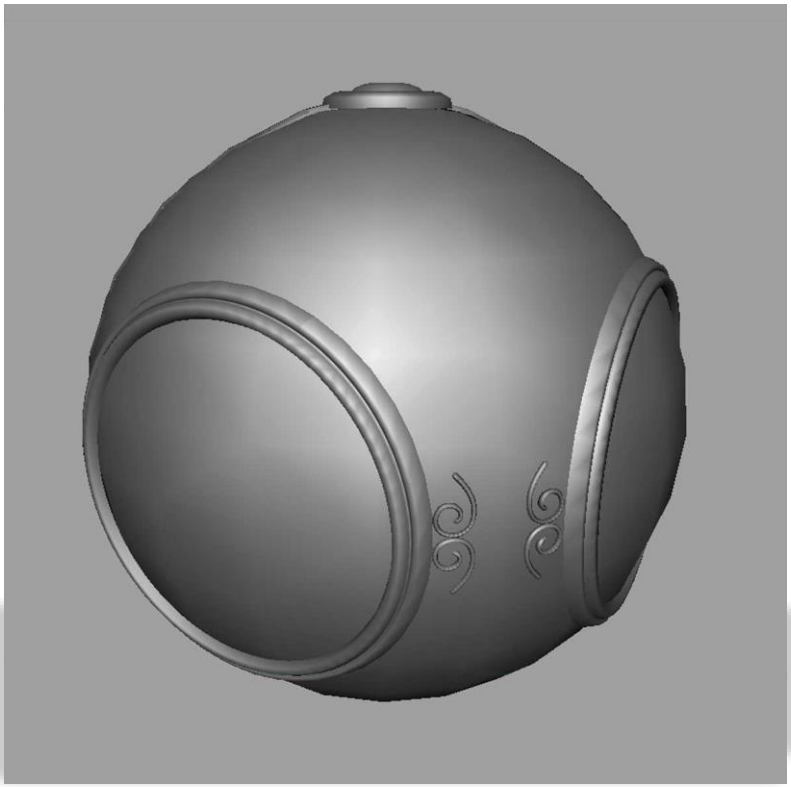
- Texture Mapping
  - Uses images to fill inside of polygons
- Bump Mapping
  - Emulates altering normal vectors during the rendering process
- Environment (reflection mapping) Mapping
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
  - Simulate complex mirror-like objects

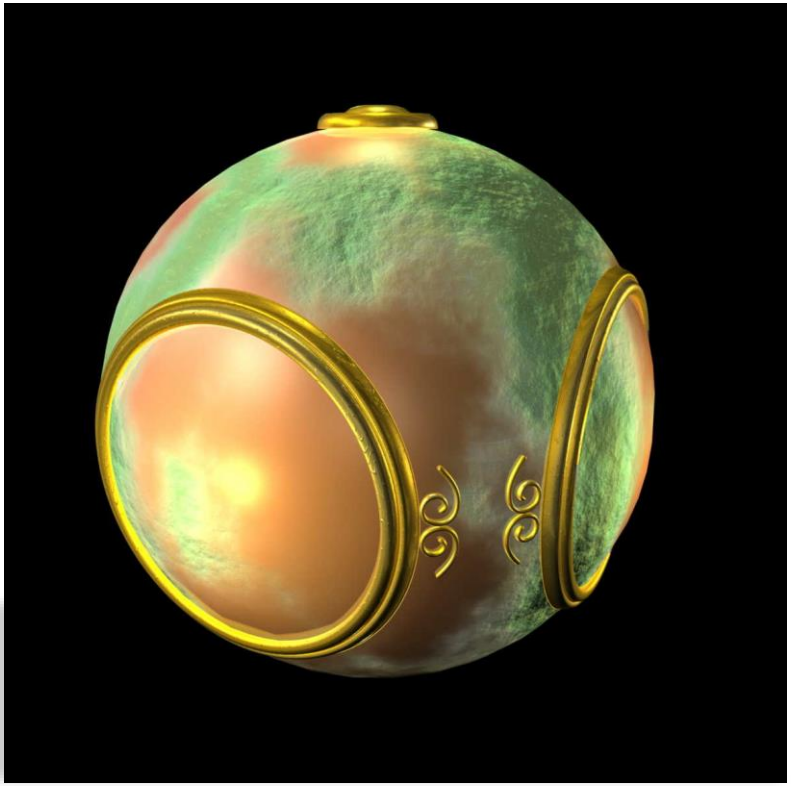
Texture Mapping



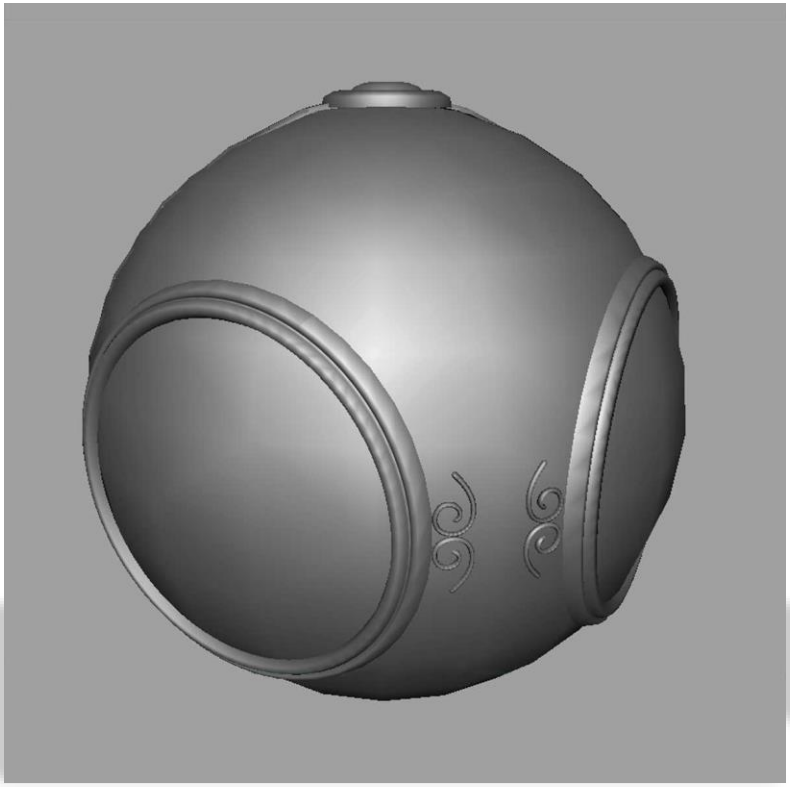


Bump Mapping





Environment Mapping

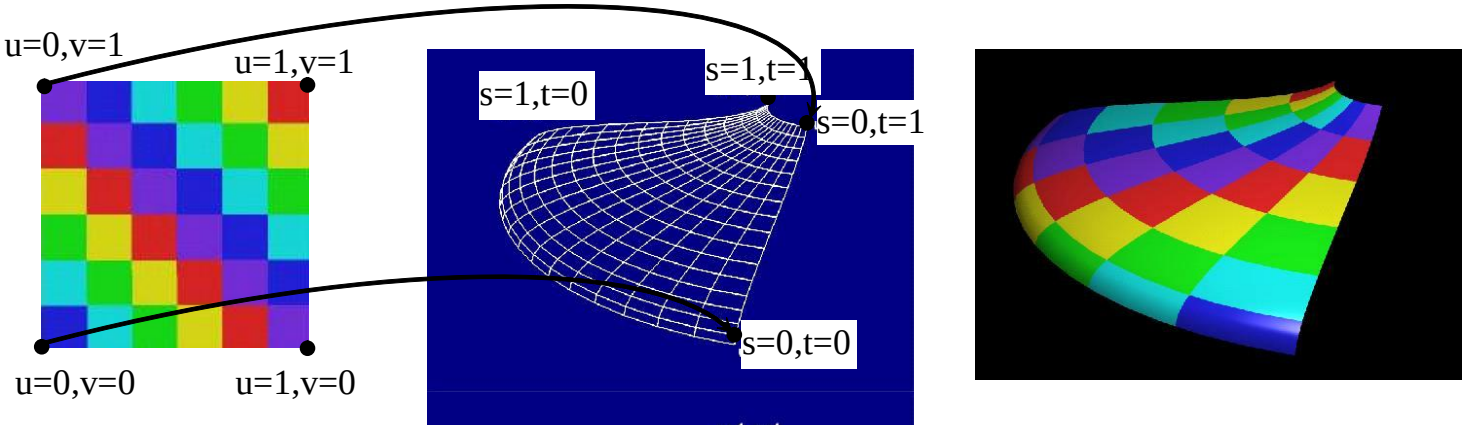


Definition of texture mapping

- Texture mapping is the process of transforming a texture on to a surface of a 3D object（adding surface detail by mapping texture patterns to the surface）.
- It is like mapping a function on to a surface in 3D
  - The domain of the function could be 1D,2D or 3D
  - The function could be represented by either an array or it could be an algebraic function.

Definition of texture mapping

- Developed by Catmull(1974),Blinn and Newell(1976) , and others



(a) 纹理空间　（b） 景物空间的曲面片　(c) 纹理映射后的曲面片

Texture usage

- To use a texture, we usually need 3 steps：
  - First，texture acquisition (获取).
  - Second，texture mapping (贴图).
  - Third，texture filtering(滤波).

1. Texture acquisition

- The first task of texturing is texture acquisition. There are a variety of methods to generate the texture image.
  - **Manual drawing:** You could simply draw a texture by hand.
  - **Taking photographs:** You could simply take a photo of the material which you want to model.
  - **Procedure texture**(过程纹理)
  - **Texture synthesis** (纹理合成)

Procedure texture

- Synthesis textures by writing special procedures which simulate either physical formation process or simply the appearance of material.
- For example, certain patterns such as marble or wood, they can be easily simulated by very simple functions.

Procedure texture

- Advantages:
  - They can be very compact, as they only need to store the procedure itself and some parameters.
  - By changing the parameters of the procedures, you can easily change the appearance of the materials, providing excellent controllability.

- Disadvantages
  - However, procedural synthesis can only be applied to some specific class of textures. – For a texture that has no known procedural code, it is not able to synthesis it.

Procedure texture

- The most popular procedure synthesis method : Perlin Noise
- From 1985, Perlin Noise has been widely adopted as the standard for procedural texture generation.
- The basic idea of Perlin noise is simple and elegant.

- **Perlin Noise**

$$perlin = \sum_{i=0}^{n-1} interpolate(white_i) \times p^i$$

  - **In the above equation, white noise at band *i* is simply a white noise with specific image size, has size 2^i. Because different bands have different sizes, we need to interpolate(插值) between them before summation.**
  - **Persistence is a user-specified parameter, which simply controls the relative weight of different frequency bands. Usually, it is within [0,1].**

Procedure texture

- **Perlin Noise**

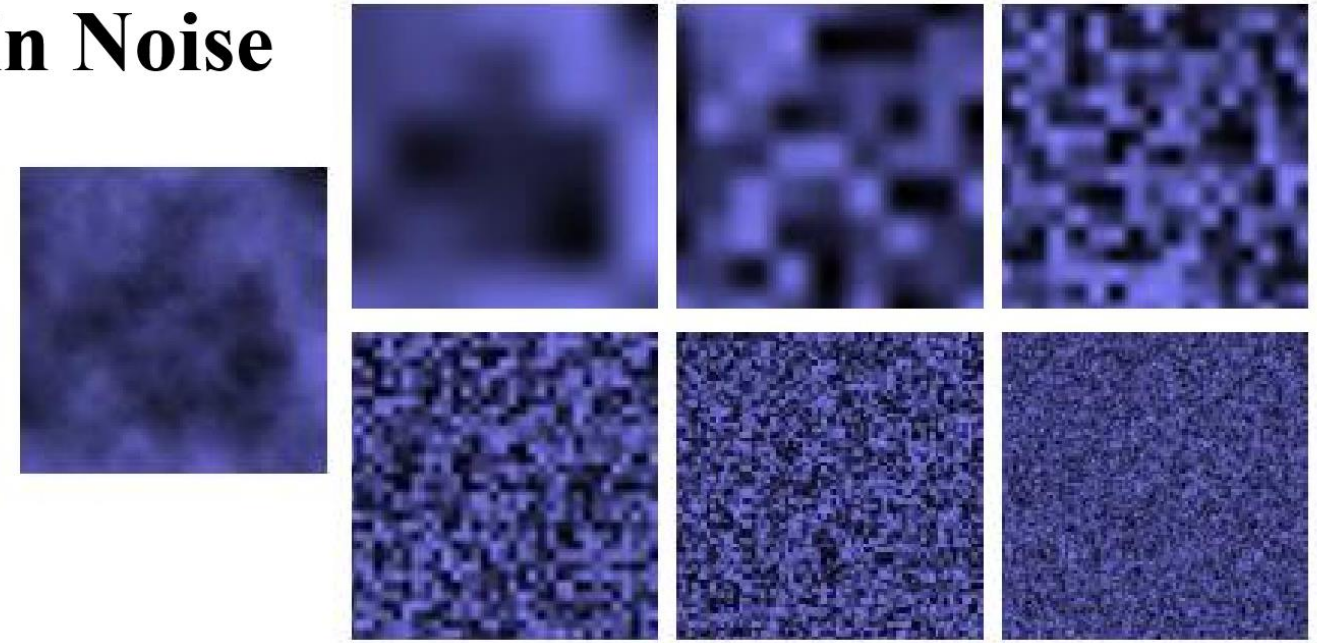


Figure 1: *2D Perlin noise example. Left: the Perlin noise image. Right: the individual noise bands, from low to high frequencies. Image courtesy of [Elias 2003].*

Procedure texture

- **Perlin Noise**
  - **Based on Perlin noise, a variety of textures can be synthesized by proper procedures, such as marble and wood. Their formulas are as follows:**

$$
\begin{aligned}
marble &= cosine(x + perlin(x, y, z)) \\
g &= perlin(x, y, z) * scale \\
wood &= g - int(g)
\end{aligned}
$$

Texture Synthesis

- What is texture synthesis?
  – An alternative approach is to synthesize a new texture from a given example. This is certainly more user friendly, because to use the algorithm, all you need to do is to provide an image sample.
  – As much research has been devoted to texture analysis in both the computer vision and graphics community, so we have many practical algorithms.

Texture Synthesis

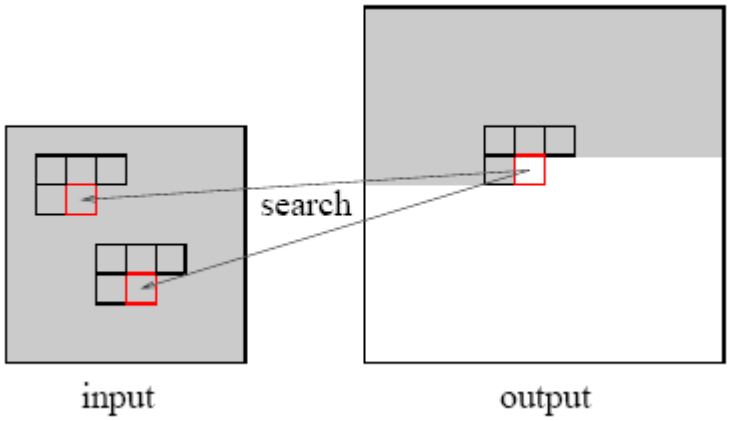
- In computer graphics, texture synthesis techniques could be classified into 2 classes:
  - Pixel-based texture synthesis
  - Patch-based texture synthesis

Texture Synthesis

- Pixel-based texture synthesis

  – Synthesis a new texture pixel by pixel, where the value of each pixel is determined by the local neighbourhood(e.g. 3*3, 5*5), choosing the input pixel with a similar neighborhood

  – These works include: “Texture Synthesis by Non-parametric Sampling” [Efros99] ， “Fast texture synthesis using tree-structured vector quantization” [Wei and Levoy00]



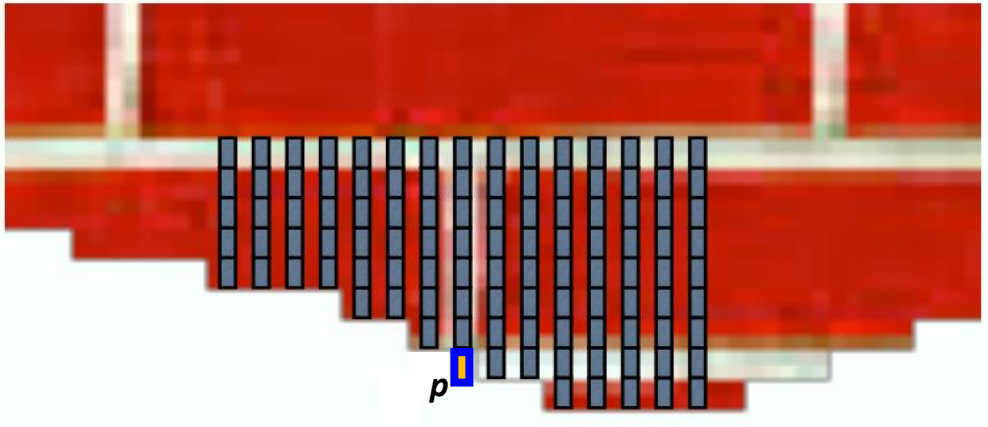
Pixel-based texture synthesis

Grow one pixel at a time

- How to paint this pixel?

Pixel-based texture synthesis

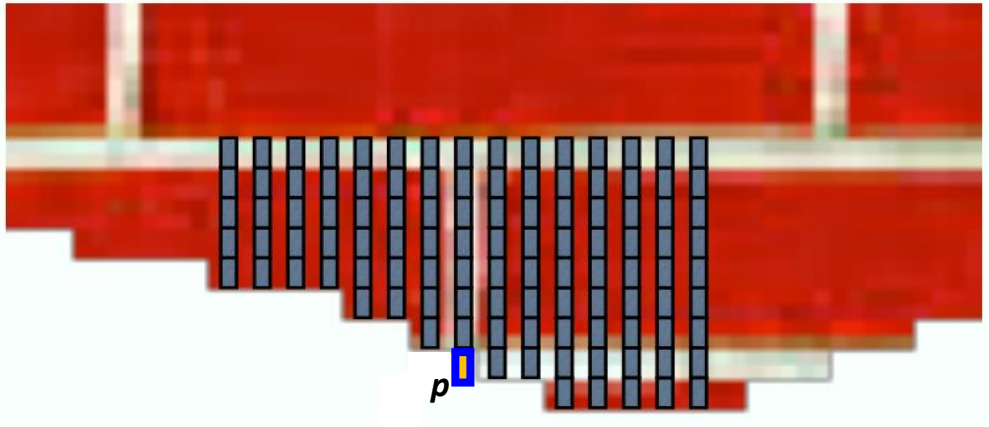
Grow one pixel at a time



- How to paint this pixel?
- **Ask neighbors**

Pixel-based texture synthesis

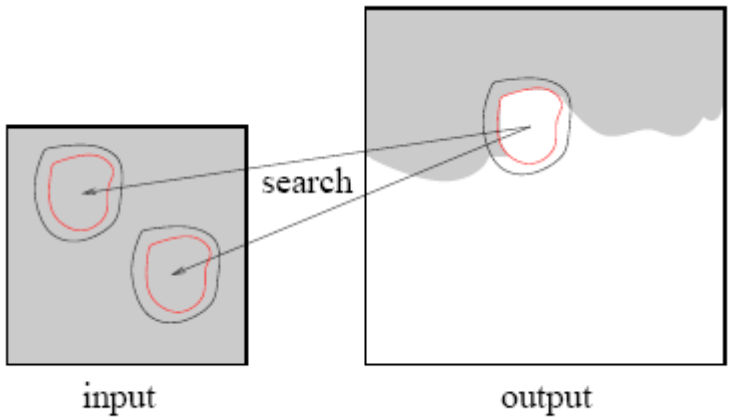
Grow one pixel at a time



- How to paint this pixel?
- **Ask neighbors**
  - Markov assumption (locality)
  - Value is not fixed (randomness)
  - Value of p: a conditional probability distribution on neighbors

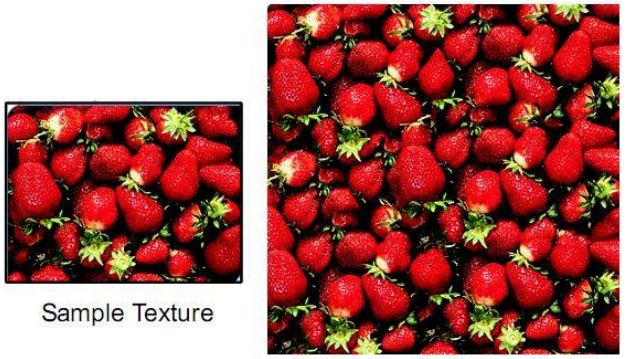
Texture Synthesis

- Patch-based texture synthesis
  - Pixel-based approaches could be improved by synthesis patches rather than pixels.
  - One of the most important algorithm is Graph-Cut[Kwatra03], which offers the best quality so far.
  - The patch is chosen also by matching neighborhoods.

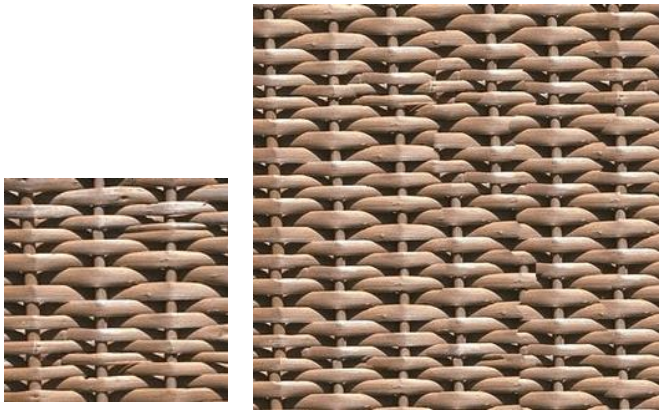


Some result of Texture Synthesis

Input Graph cut

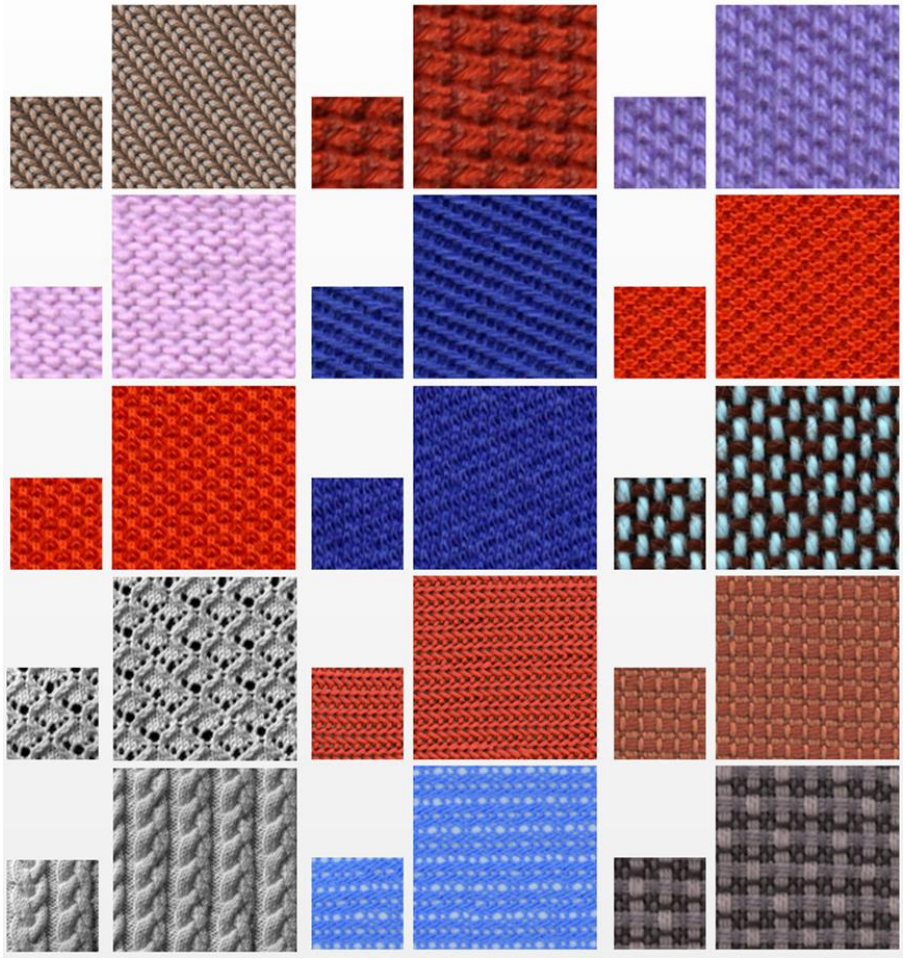


Sample Texture Synthesized Texture



Texture Synthesis

- Some result of texture synthesis of various fabrics

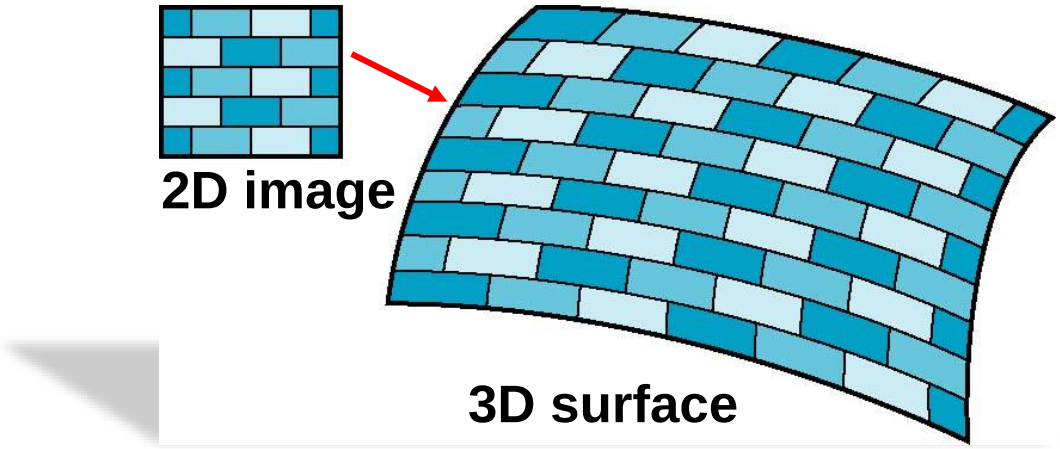


2. Texture mapping

- Given a model, and a 2D texture image.
- Map the image onto the model:
  - By a function which maps a point on the model onto (u,v) image coordinates, this function is called surface mapping function
- When shading a point on the model, we look up the appropriate pixel from the 2D texture, and use that to affect the final color.

Is it simple?

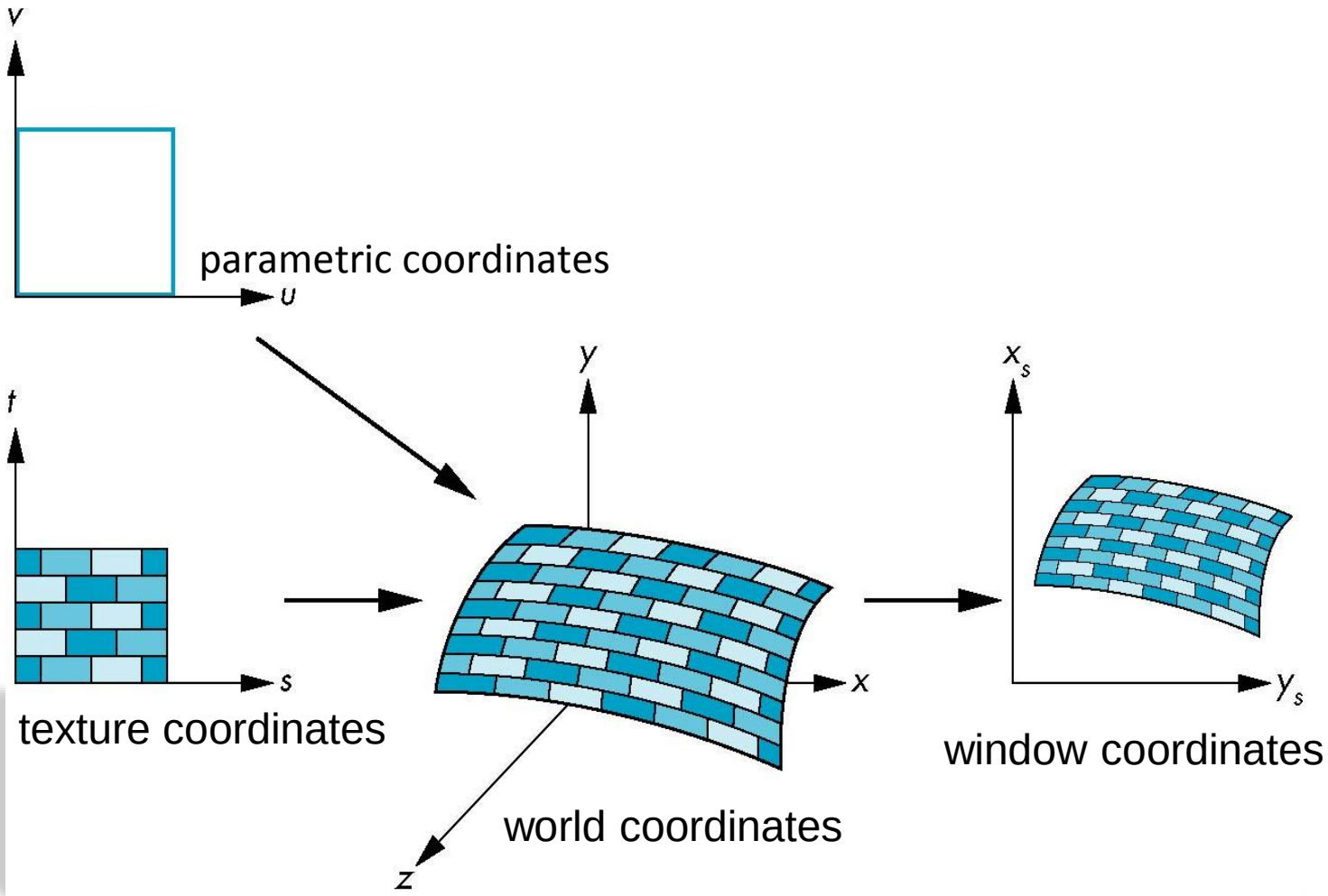
- Although the idea is simple---map an image to a surface---there are 3 or 4 coordinate systems involved



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

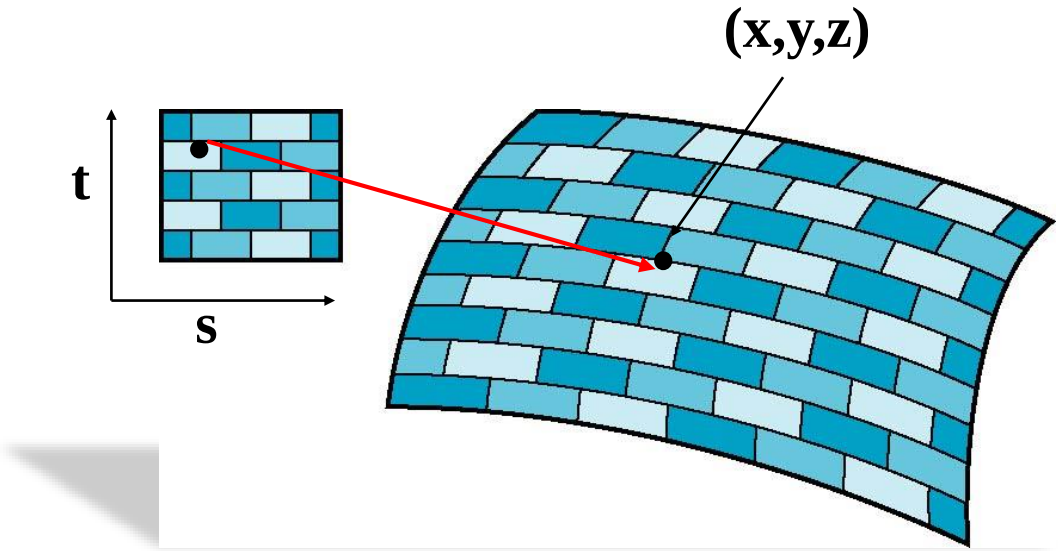
Texture Mapping



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

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



Basic Conpect

- Relate a 2D image to a 3D model
- Texture coordinates
  - ✓ Texture coordinate is a 2D coordinate(u,v) which maps to a location on a texture map
- Texture coordinate are over the interval [0,1],typically

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

Assigning Texture Coordinates

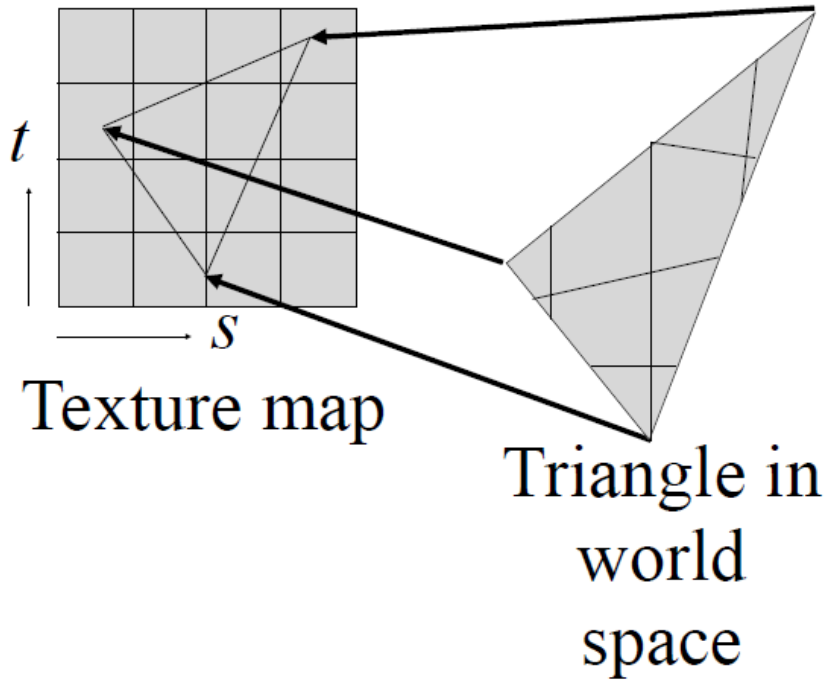
- You must provide texture coordinates for each vertex.
- The texture image itself covers a coordinate space between 0 and 1 in two dimensions usually called *u* and *v* to distinguish them from the *x*, *y* and *z* coordinates of 3D space.
- Each data element in a texture is called a texel. On screen, a texel may be mapped to
  - A single pixel
  - Part of a pixel(for small polygons)
  - Several pixels(if the texture is too small or the polygon is viewed from very close)

Assigning Texture Coordinates

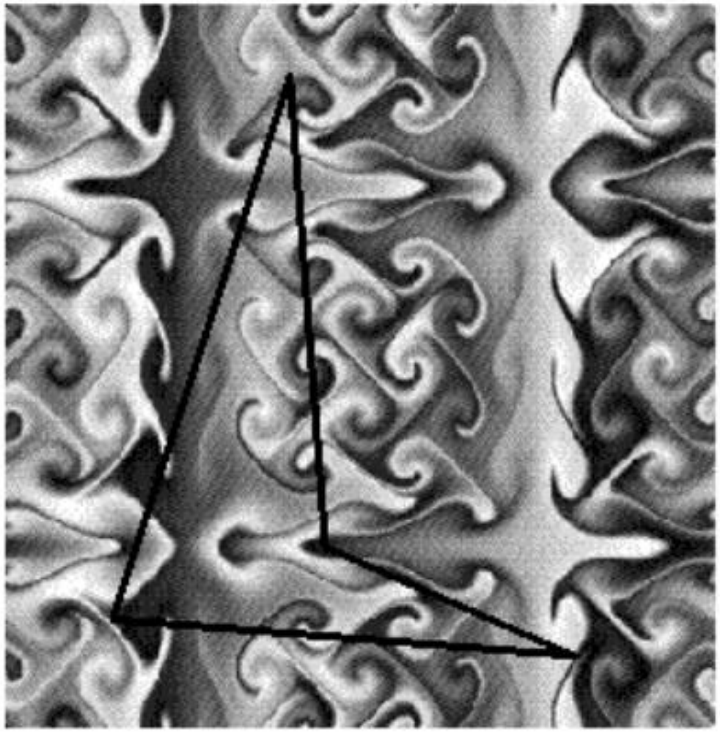
- A vertex's texture coordinates determine which texel(s) are mapped to the vertex.
- Texture coordinates for each vertex determine a portion of the texture to use on the polygon.
- The texture subset will be stretched and squeezed to fit the dimensions of the polygon.

Texture Interpolation

- Specify where the vertices in world space are mapped to in texture space
- Linearly interpolate the mapping for other points in world space
  - Straight lines in world space go to straight lines in texture space



Texture Example

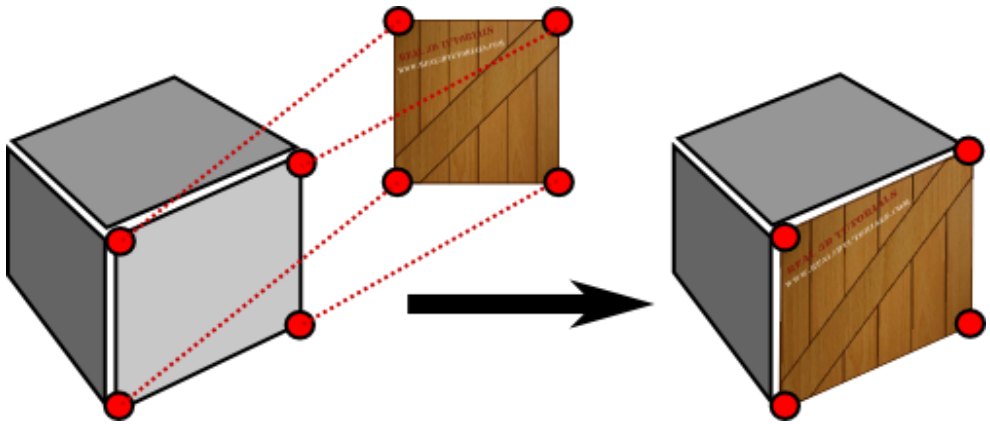


Polygonal texture mapping

- Establish correspondences
- Find compound 2D-2D mapping
- Use this mapping during polygon scan conversion to update desired attribute (e.g. color)

1.Establish Correspondences

- Usually we specify texture coordinates at each vertex
- These texture coordinates establish the required mapping between image and polygon



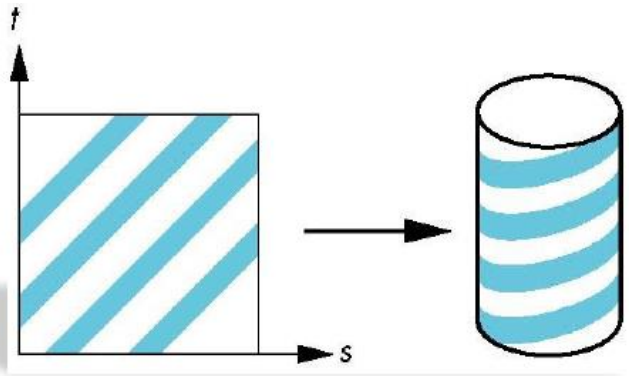
1.Establish Correspondences

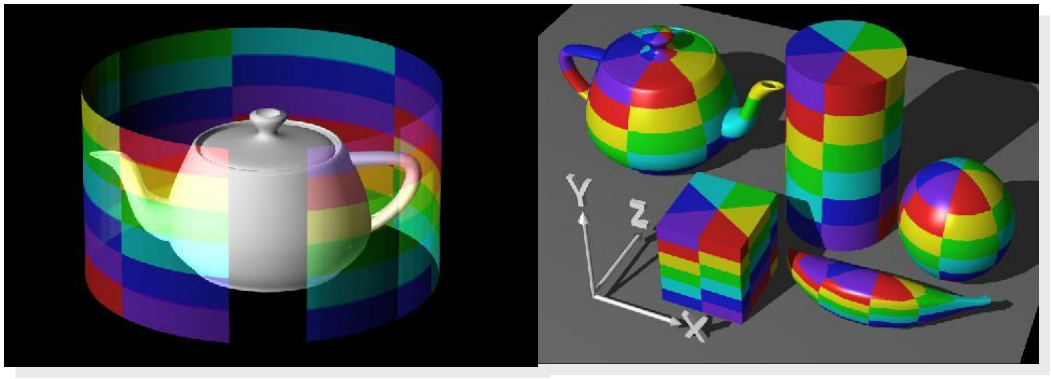
- How to specify surface mapping function?
- By natural parameterization
  - For regular objects, such as sphere, cube, cylinder
- By manually specify texture coordinates
  - For complex objects, each vertex is specified a texture coordinate

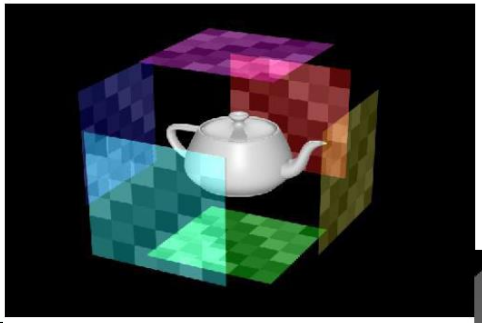
By natural parameterization

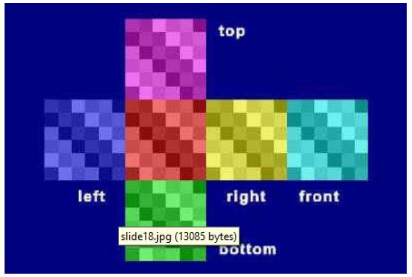
- **Natural parameterization**
  - **Sphere:**
    - **You could use spherical coordinates** $(\theta, \phi) = (\pi u, 2\pi v)$
  - **Cylinder:**
    - **You could use cylinder coordinates** $(u, \theta) = (u, 2\pi v)$
  - **Cube**
    - **Each face could directly use its face coordinates.**

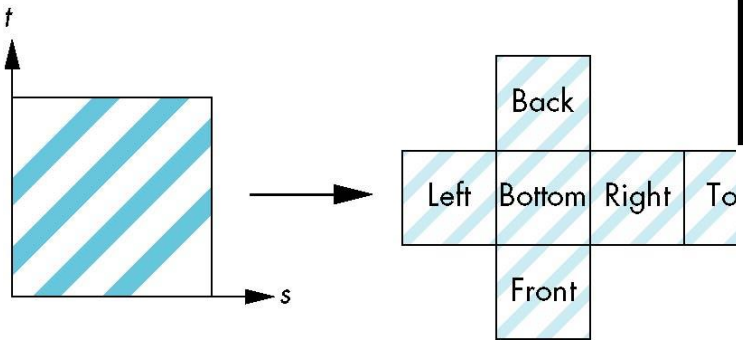
By Natural parameterization

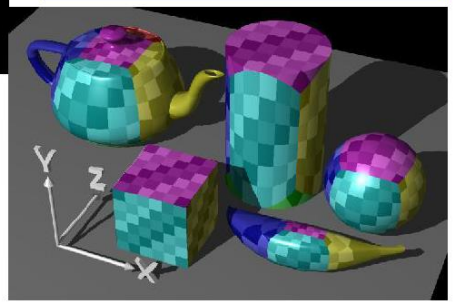








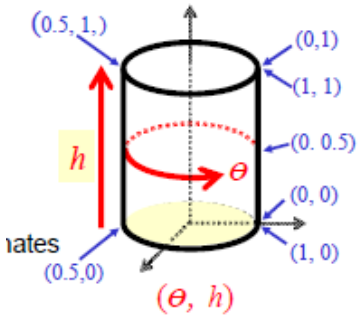




Cylindrical Mapping

- parametric cylinder

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



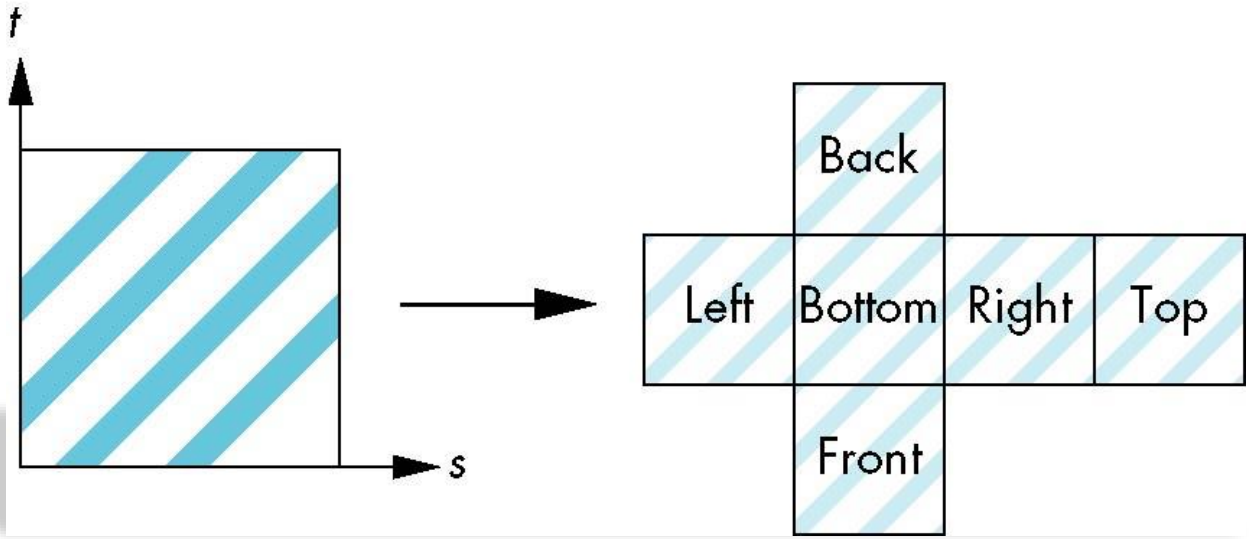
maps rectangle in u, v space to cylinder of radius r and height h in world coordinates

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

maps from texture space

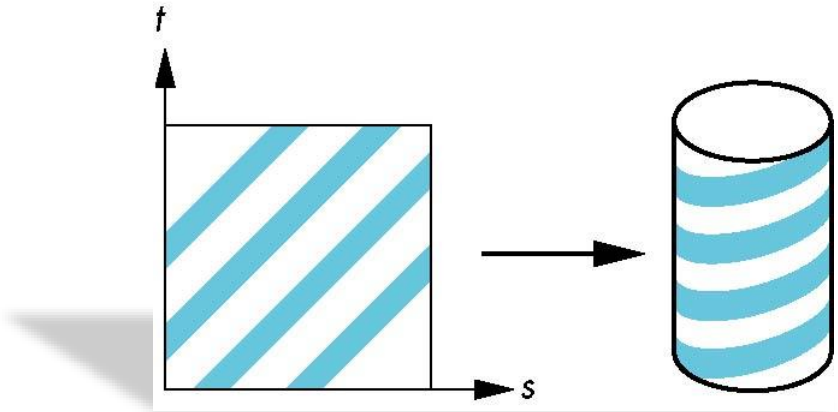
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps



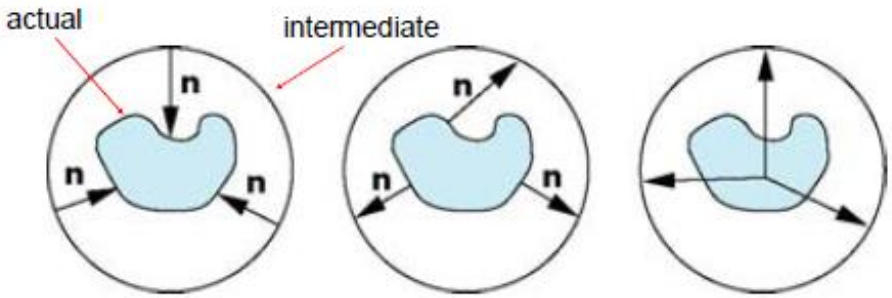
Two-part mapping

- One solution to the mapping problem is to first map the texture to a simple intermediate surface
- Example: map to cylinder



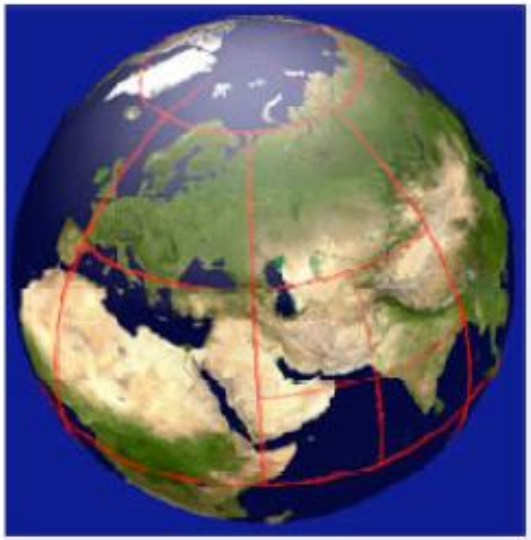
Second Mapping

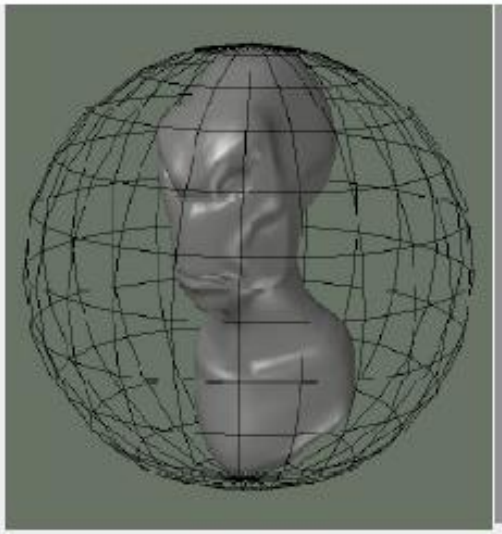
- Map from intermediate object to actual object
  - Normals from intermediate to actual
  - Normals from actual to intermediate
  - Vectors from center of intermediate



Spherical Mapping

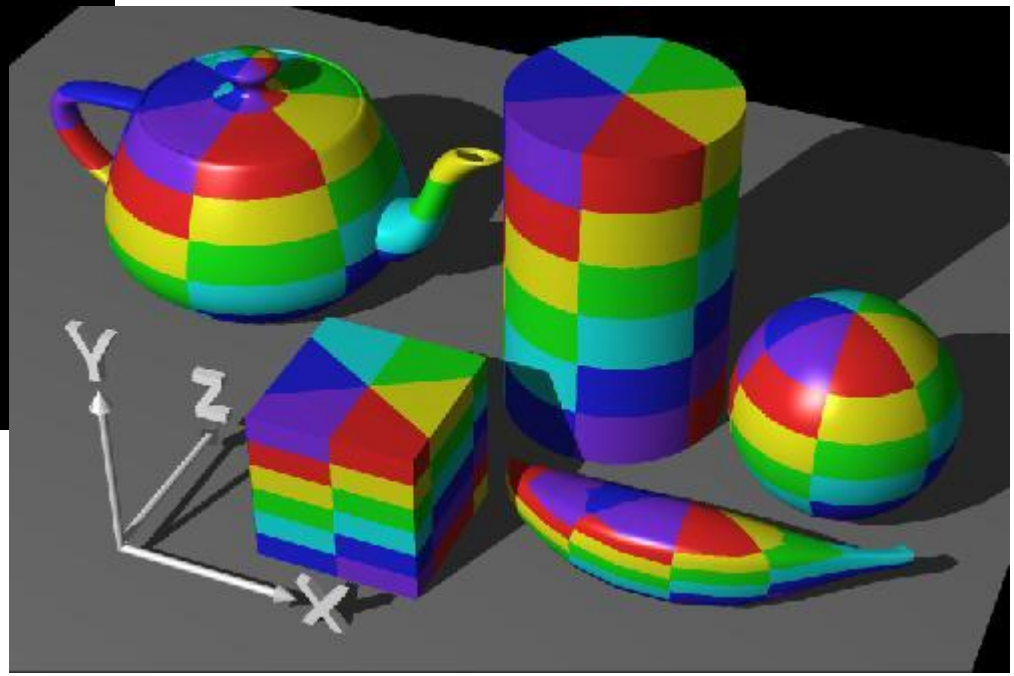
- Use, e.g., spherical coordinates for sphere
- Place object in sphere
- "shrink-wrap" sphere to object





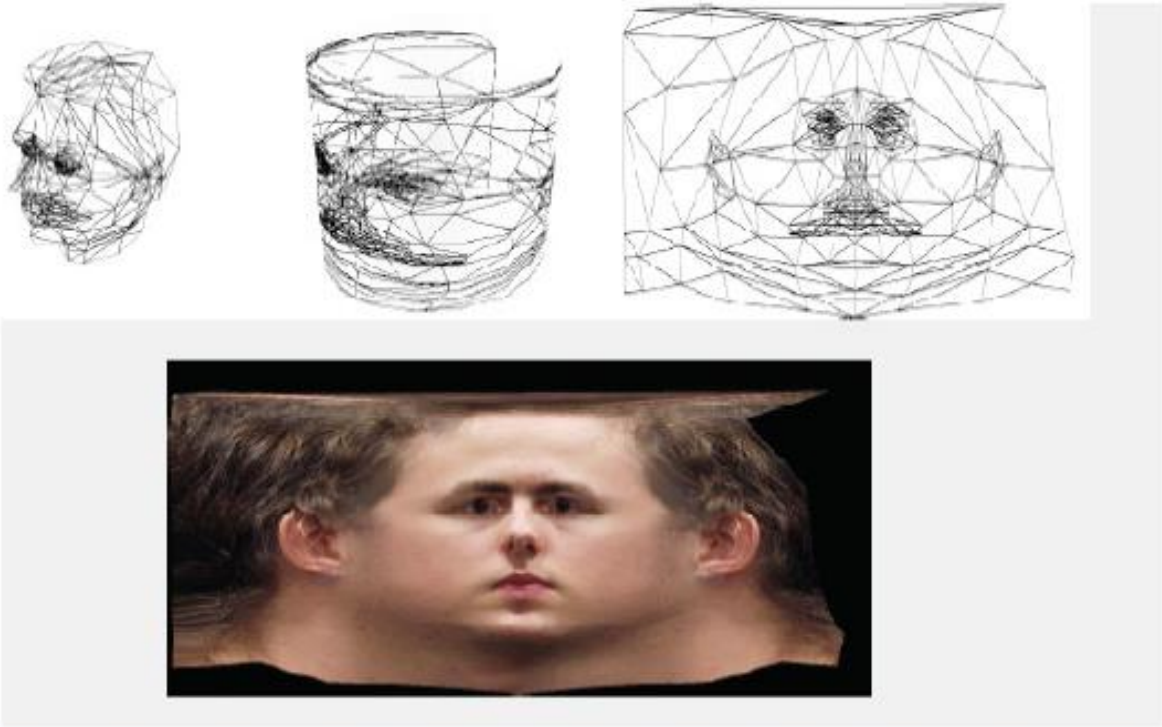
Cylindrical Mapping

- Cylinder: r, θ, z with (s,t) = (θ/(2π),z)
  - ✓ Note seams when wrapping around (θ = 0 or 2π)

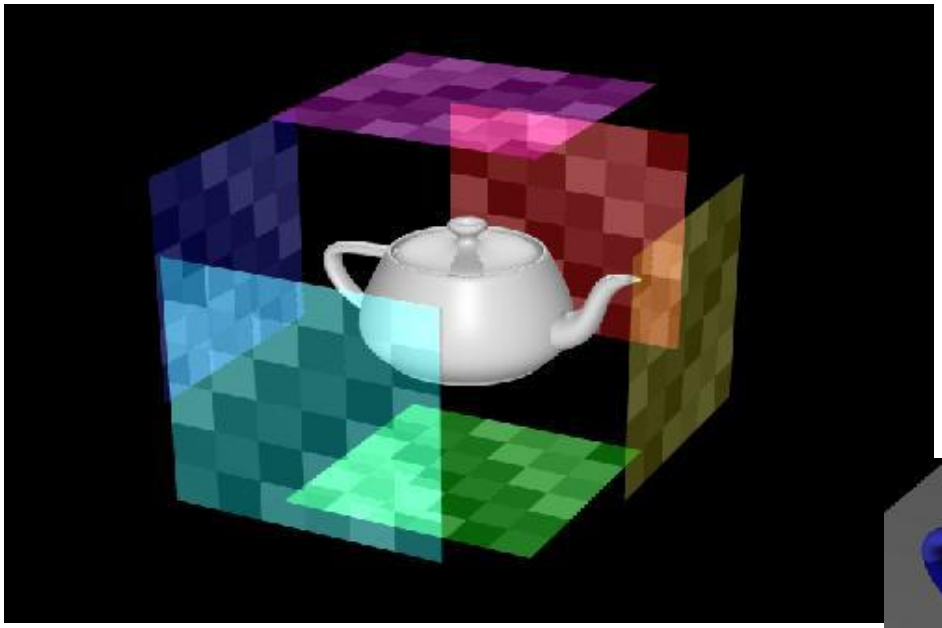


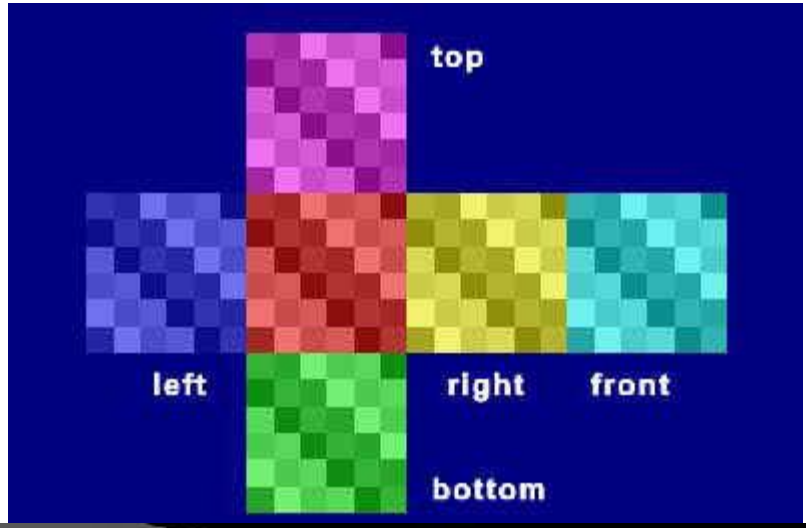
Cylindrical Mapping

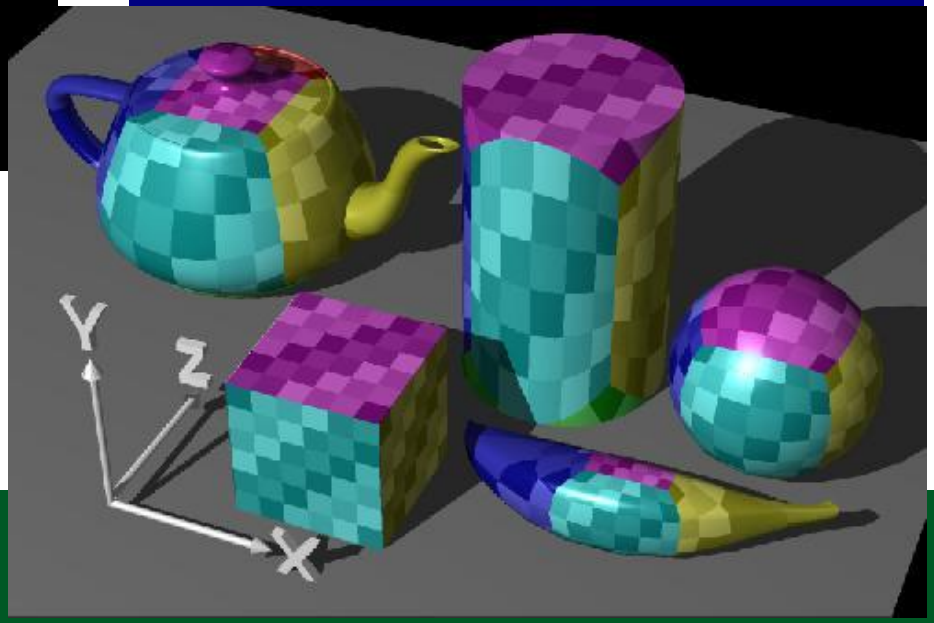
- Similar as spherical mapping, but with cylilnder
- Useful for faces



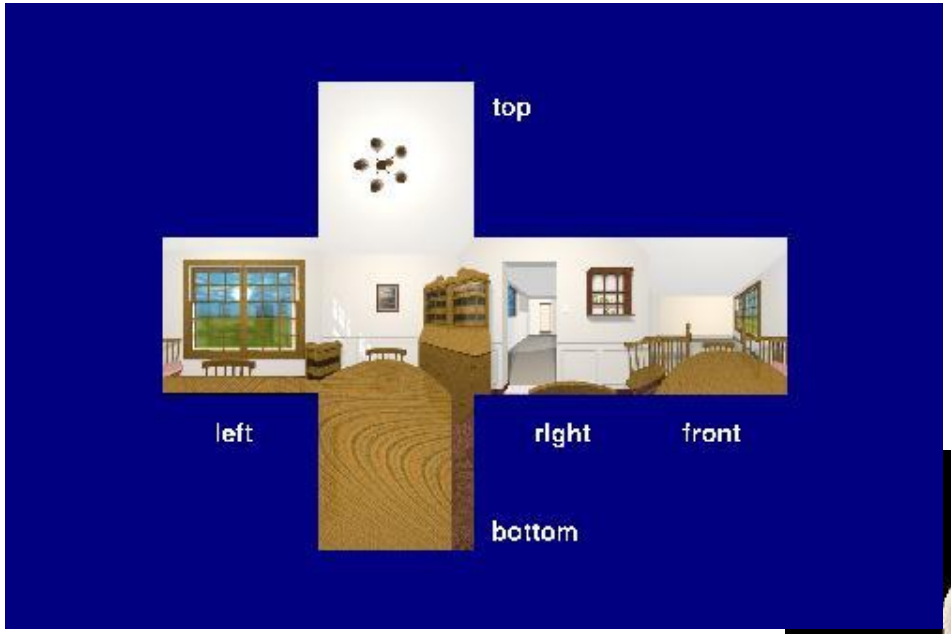
Cube Mapping





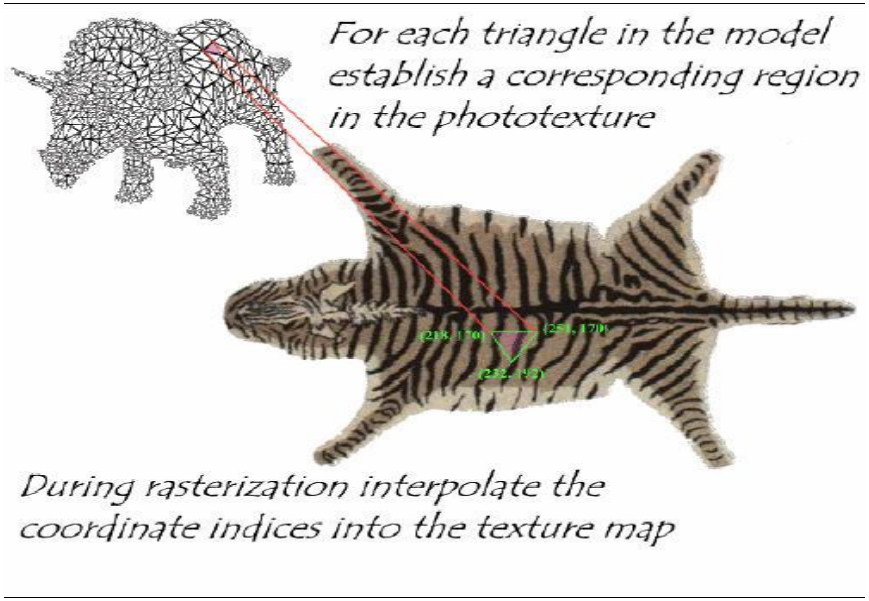


Cube Mapping

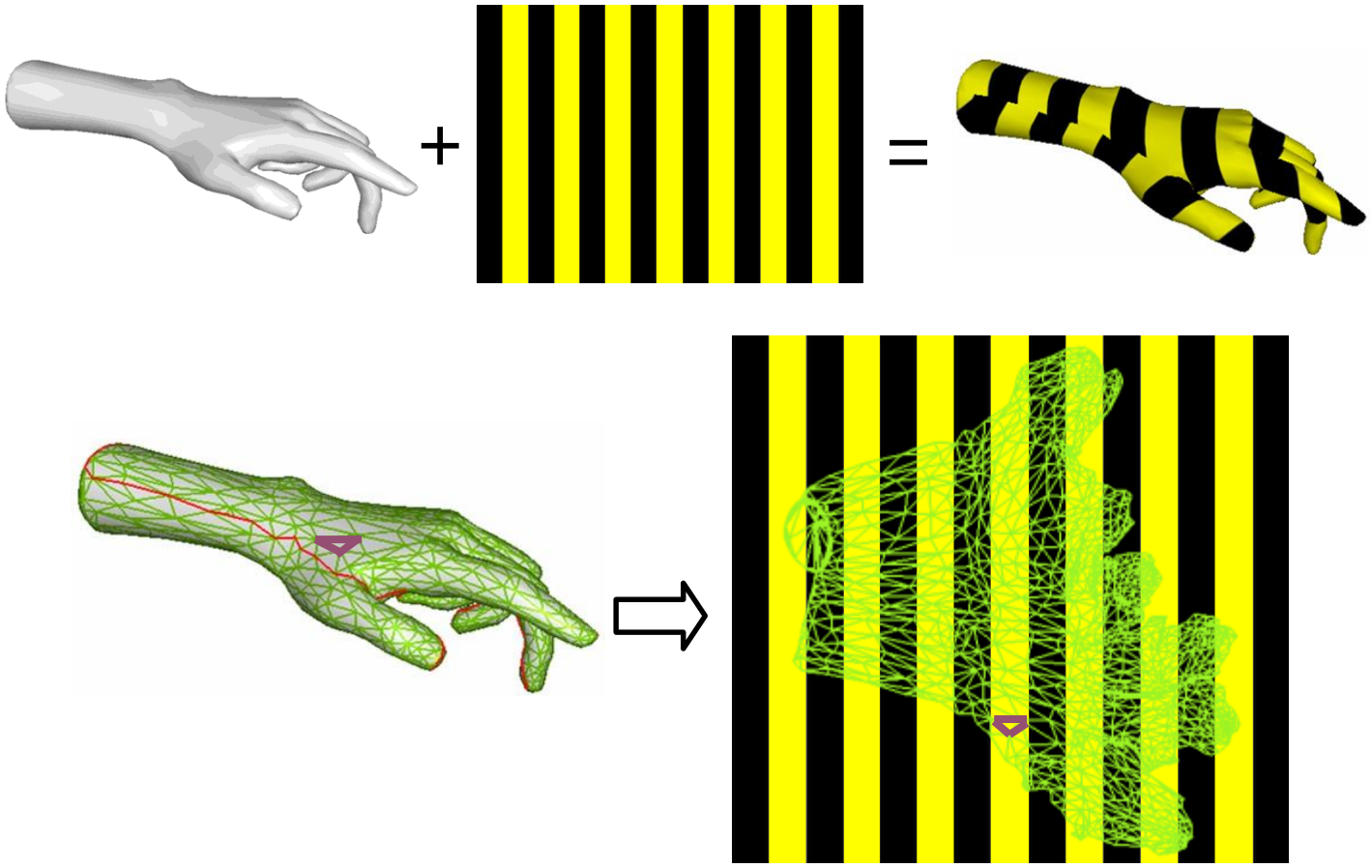


By Manually Specify texture coordinates

- Manually Specify texture coordinates
  - Each vertex is specified a texture coordinates
  - Mapping a triangle in image space to object space



By Manually Specify texture coordinates



Find compound 2D-2D mapping

- Since the texture is finally seen on screen which is 2D, it makes sense to combine two mappings (from image to 3D space and then from 3D to screen space) into single 2D-2D mapping
- This avoids texture calculations in 3D completely
- This simplifies hardware implementation of graphics pipeline.

Attributes modulated for texture

- *Surface color* **(diffuse reflection coefficients)**
  - –most commonly used parameter for texture mapping. It is used like wrapping an image on to a surface, like a label on a bottle.
- *Specular and diffuse reflection* **(environment mapping)**
  - –used to capture reflection of the environment on to a surface. Commonly used in exhibiting shiny metallic surfaces.
- *Normal vector perturbations* **(bump mapping)**
  - –used to generate a rough surfaces like that of an orange.

Crude texture mapping code

```
// for each vertex we have x,y,z and u,v
for (x=xleft; x < xright; x++) {
    if (z < zbuffer[x][y] ){
        z[x][y] = z;
        raster[x][y] = texture[u][v]; // replacing color
    }
    z=z+dz;
    u=u+dv;
    v=v+dv;
}
```

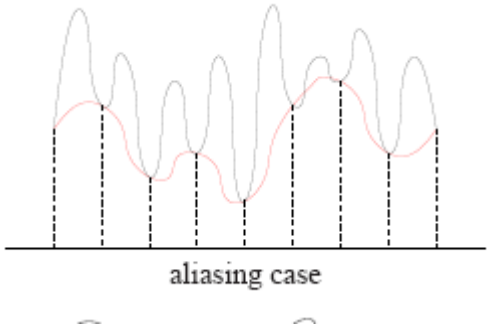
- Note that instead of replacing color as done in code, you can also modulate the color
- Instead of color, you can of course choose to modify some other property of surface

3. Texture Filtering

- Now we know how to generate texture and how to map it onto an object. The next step is to render it.
- However, if you just do this directly, you would encounter some artifacts. These artifacts are caused by signal aliasing(走样)

Texture Filtering

- Aliasing(走样) and Anti-aliasing(反走样)
  - – Generally, aliasing is caused when a signal is sampled at too low frequency rate. So that many high frequency features of the signal are missed.
  - – e.g.: The black dash is the original signal, the red curve is the aliased signal.



Texture Filtering

- To resolve this problem, a common method is to sample at a higher rate. However, this is not always feasible. For example, in graphics applications, the screen resolution limits sample rate.
- An alternative method would be to prefilter the signal into a lower frequency one.

Isotropic(各向同性) filtering

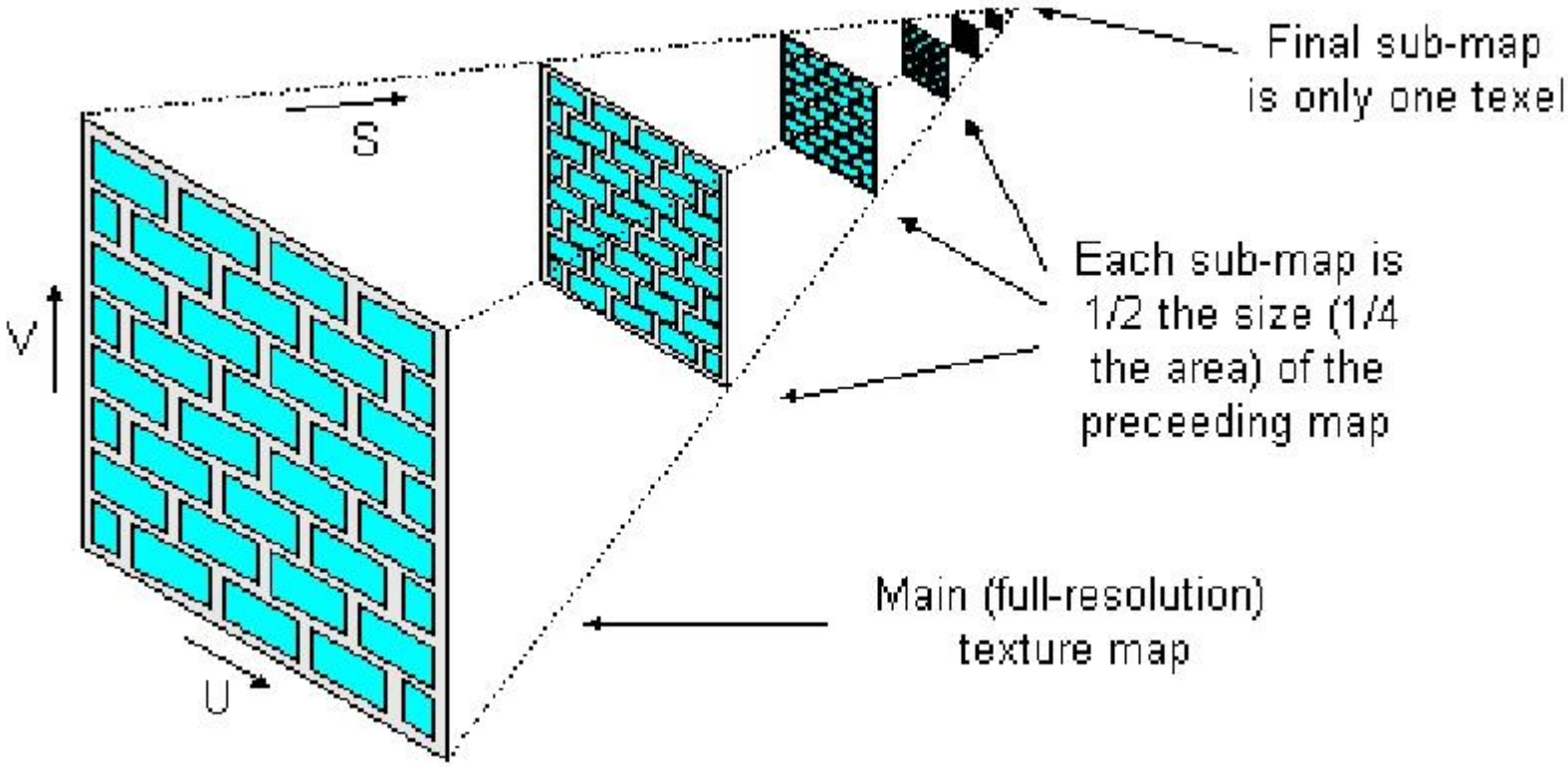
- Since most of the times, textures are known a priori, we can create various levels of these prefiltered textures in a preprocess.
  - Construct a pyramid of images that are prefiltered and re-sampled at 1/2, 1/4, 1/8, etc., of the original image's sampling
  - During rasterization, we compute the index of the image that is sampled at a rate closest to the density of our desired sampling rate.
  - This is known as mipmapping (分级细化贴图)

MipMapping

- Like any other object, a texture mapped object can be viewed from many distances.
- Sometimes, that causes problems.
  - A low-resolution texture (say, 32x32) applied to a big polygon (say, one that occupies a 512x512 area on screen) will appear blocky.
  - Conversely, if you apply a high-resolution texture to a small polygon, how do you decide which texels to show and which to ignore?

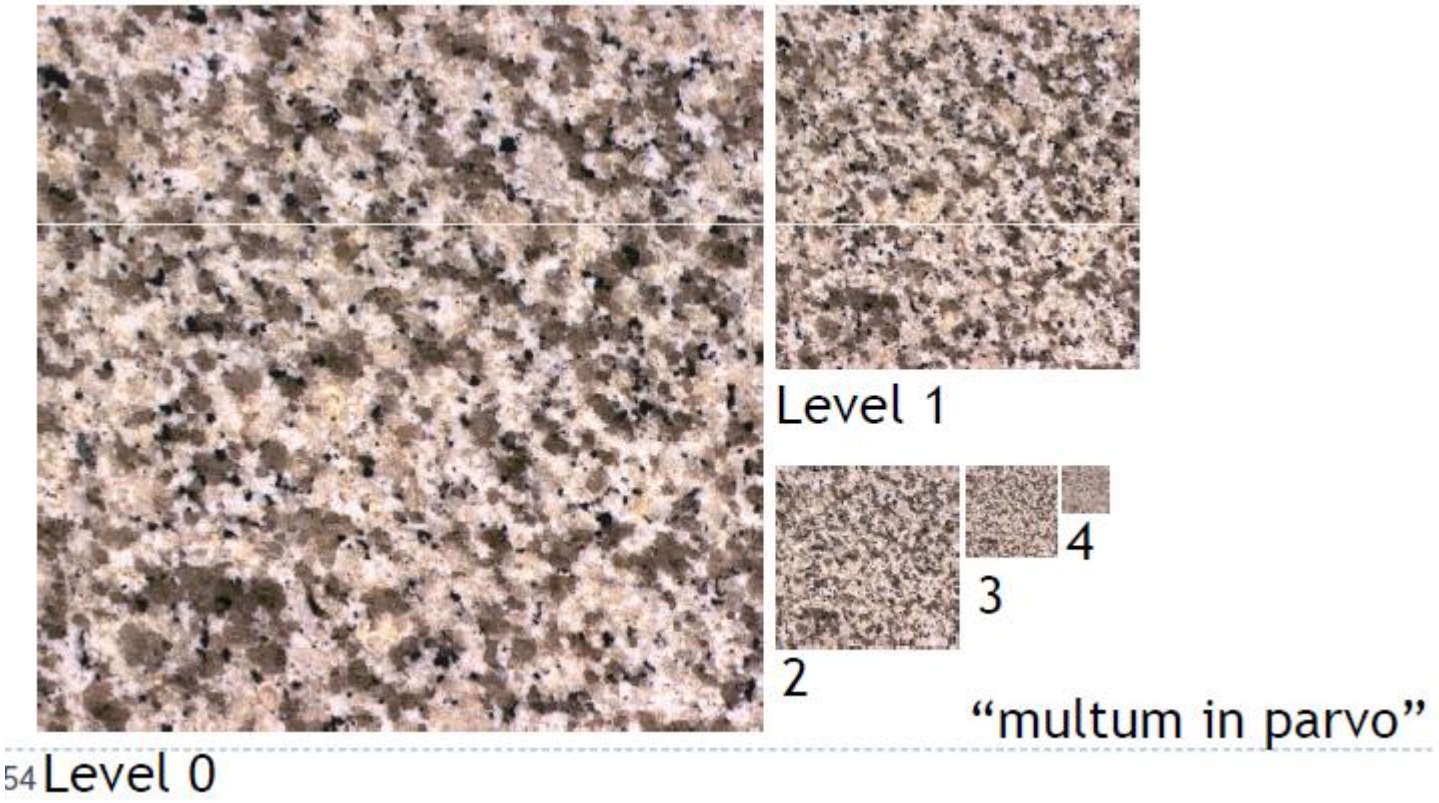
Mipmapping

- The basic idea is to construct a pyramid of images that are prefiltered and down sampled.



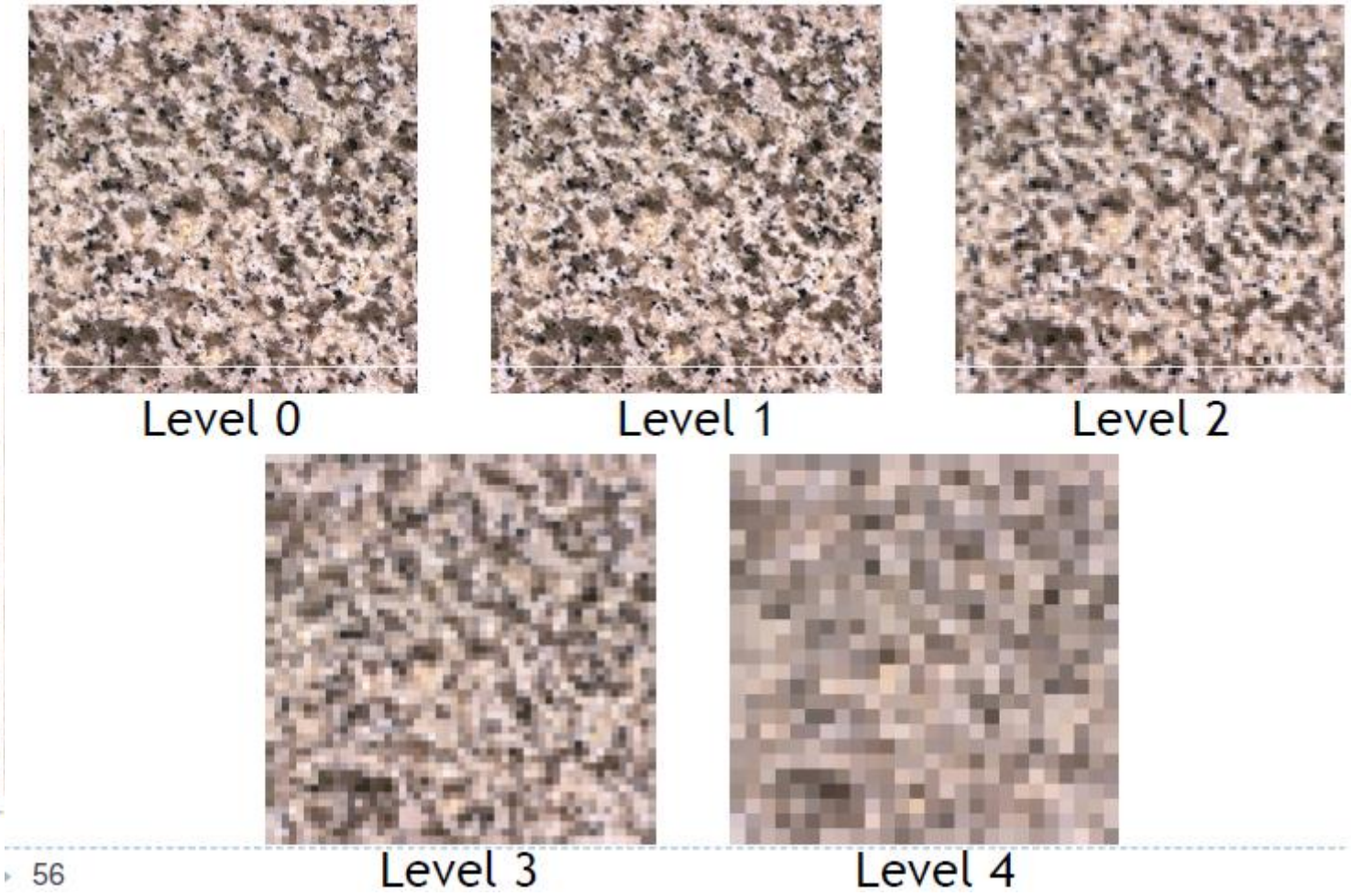
Mipmapping (continued)

Example: resolutions 512x512, 256x256, 128x128, 64x64, 32x32 pixels.

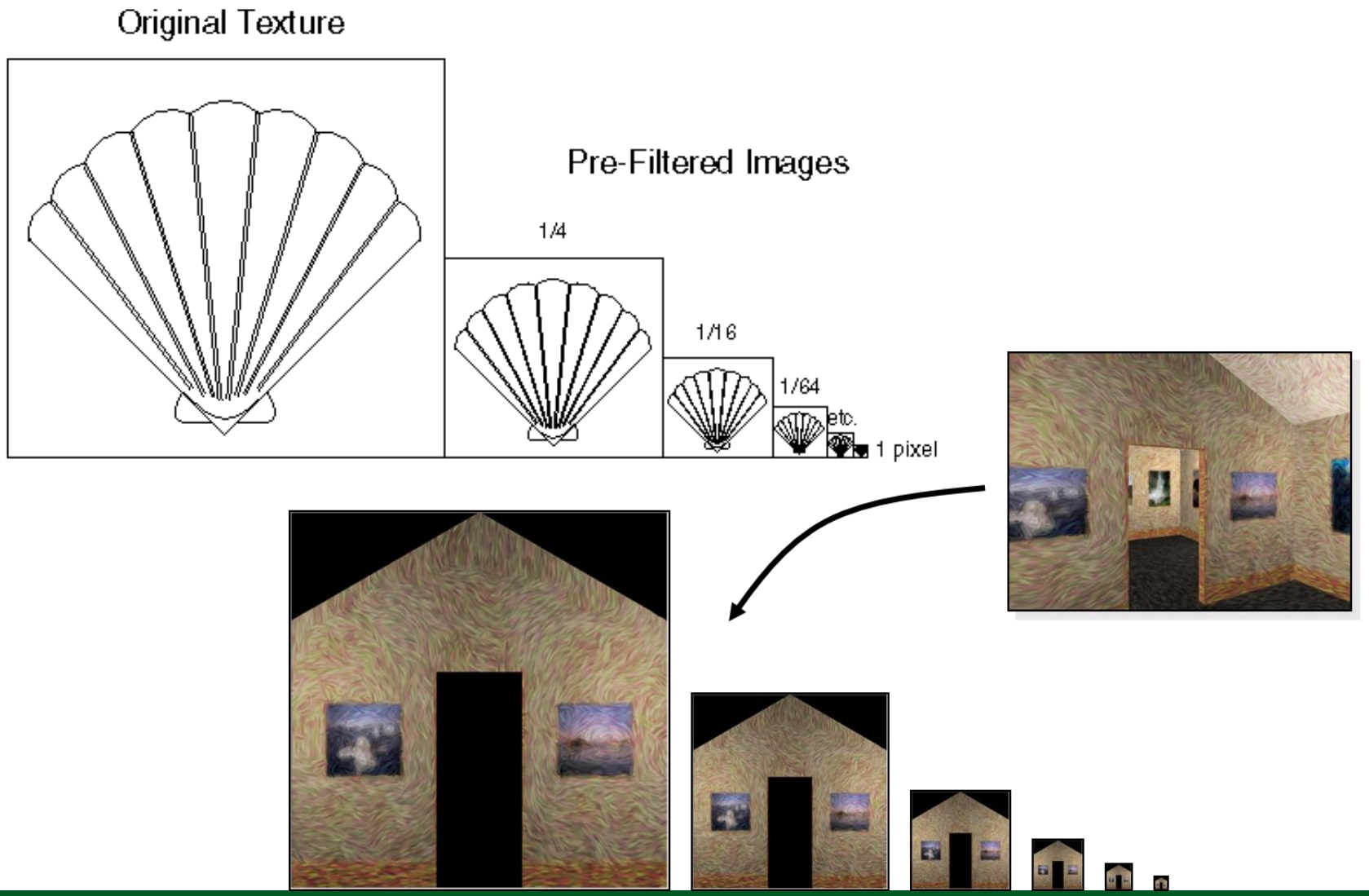


Mipmapping (continued)

- One texel in level 4 is the average of 256 texels in level 0



MipMapping Example



MIP-map Example

- No filtering:

AAAAAAAGH
MY EYES ARE BURNING

- MIP-map texturing:

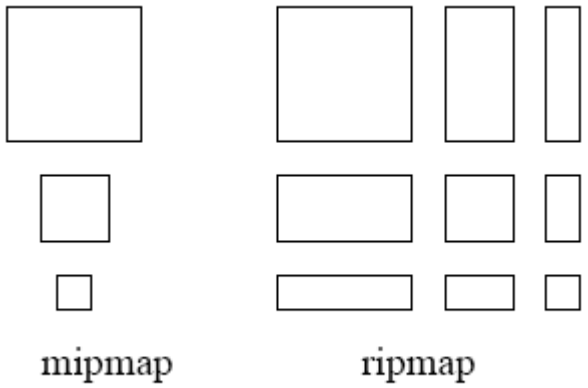
Where are my glasses?

Anisotropic(各向异性) Filtering

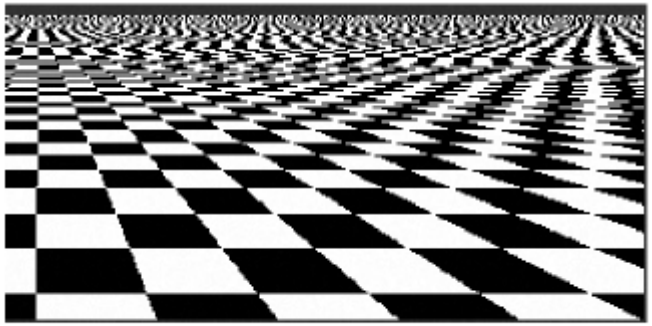
- However, one problem with mipmapping is that it can sometimes over-blur, as shown in the figure (last slide). When we build a mipmap, we always filter a higher resolution level isotropically. This can cause over-blurring when the desired filter is anisotropic (when viewed perspective). We will have to enclose it with a big isotropic footprint in order to avoid aliasing. However, this would also cause over blurring in the short axis of the anisotropic footprint.

Anisotropic(各向异性) Filtering

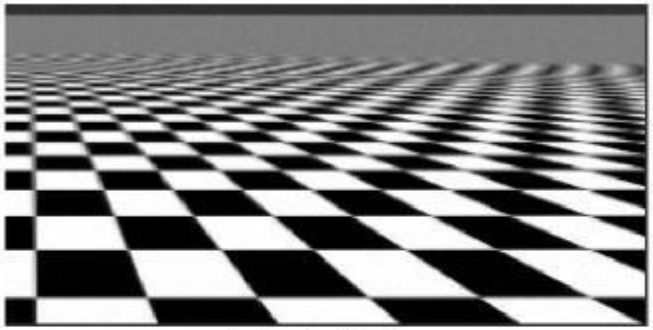
- Build a ripmap instead of mipmap by precomputing the anisotropic filtering.



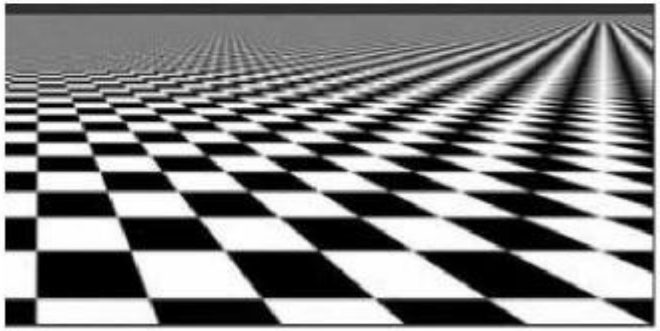
Anisotropic(各向异性) Filtering



aliasing



isotropic filtering



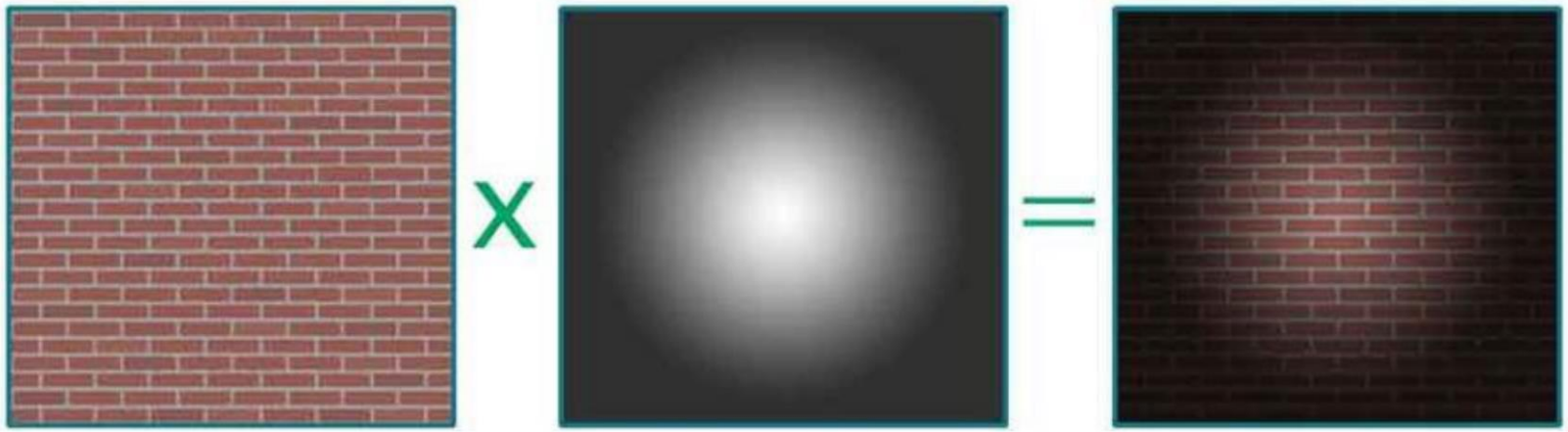
anisotropic filtering

Texture Mapping Applications

- Light maps
- Bump mapping
- Displacement mapping
- Illumination or Environment Mapping
- And many more

Light Maps

- Instead of wallpapering a polygon with the texture's colors, we can blend the texture with the existing colors.
- A "light map" is such a black-and-white texture; white texels will cause the underlying pixels to appear brighter and shinier, and vice versa.



diffuse object color
(color texture)

light map
(light texture)

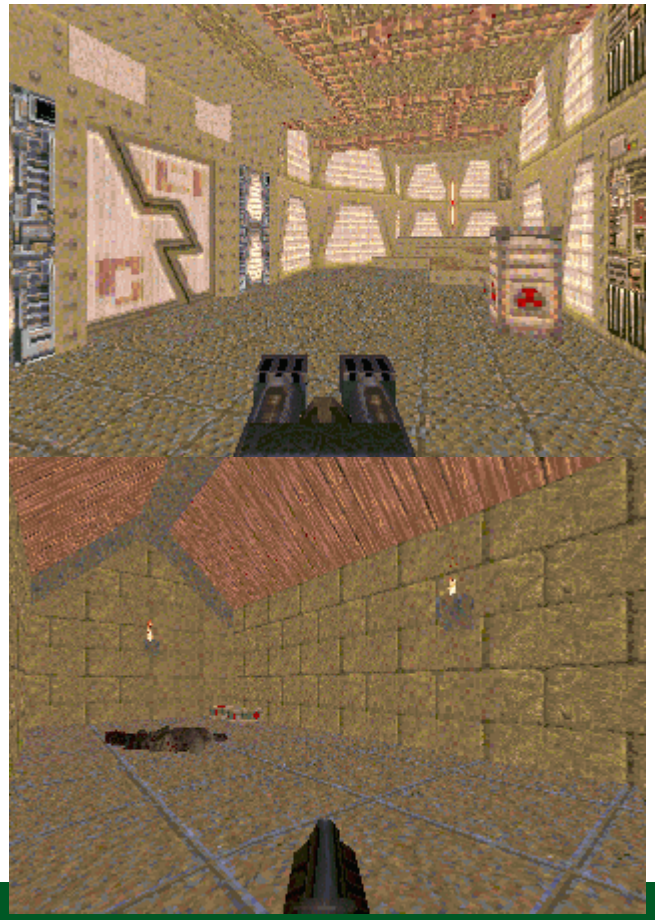
diffuse reflection component

Light Maps

- Light maps are used to store pre-computed illumination

Light map image by Nick Chirkov

Texture only



Texture map+ light map

Advanced Topics in Texture

Map more than colors

- Basic texture mapping applies colors to surfaces
  - but the method is much more general than this
  - can be used to map wide range of data onto surfaces
- We will discuss a couple of examples today
  - Bump mapping
  - Displacement mapping

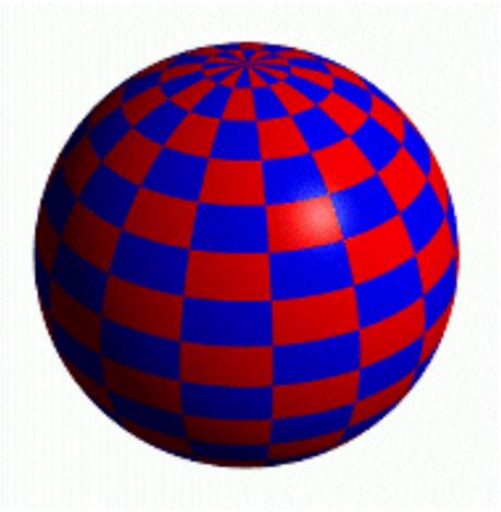
Advanced Topics in Texture

Map more than colors

- Using textures to affect theses parameters
  - Surface color - common textures
  - Surface normal - bump mapping (Blinn 1978)
  - Geometry - displacement mapping
  - Light source radiance – environment mapping(Blinn 1978)

Bump Mapping (凹凸贴图)

- Use textures to alter the surface normal
  - ✓It does not change the actual shape of the surface
  - ✓Just shaded as if it were a different shape
  - ✓ Bump Mapping assumes that the illumination model is applied at every pixel(as in Phong Shading or ray tracing).



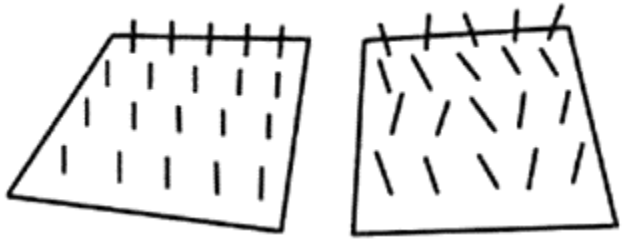
Sphere w/Diffuse Texture

Swirly Bump Map

Sphere w/Diffuse Texture & Bump Map

Finding Bump Maps

- The normal at any point on a surface characterizes the orientation of the surface at that point.
- If we perturb the normal at each point on the surface by a small amount, then we create a surface with small variations in its shape.



如果在生成图像时进行这种扰动，那么就会从光滑的模型得到具有复杂表面模型的图像。

Method of perturbation (1)

- **We can perturb the normals in many ways.**
- **The following procedure for parametric surfaces is an efficient one.**
  - **Letp(u, v)be a point on a parametric surface and the partial derivatives at the point:**

$$\mathbf{p}_u = \begin{bmatrix} \frac{\partial x}{\partial u} \\ \frac{\partial y}{\partial u} \\ \frac{\partial z}{\partial u} \end{bmatrix}, \qquad \mathbf{p}_v = \begin{bmatrix} \frac{\partial x}{\partial v} \\ \frac{\partial y}{\partial v} \\ \frac{\partial z}{\partial v} \end{bmatrix}$$

  - **The unit normal at that point:**

$$\mathbf{n} = \frac{\mathbf{p}_u \times \mathbf{p}_v}{|\mathbf{p}_u \times \mathbf{p}_v|}$$

Method of perturbation (2)

- **Displace the surface in the normal direction by a function called the bump,or displacement function *d(u, v)*, The displaced surface:**

  $\mathbf{p}' = \mathbf{p} + d(u, v)\mathbf{n}$

- **The normal at the perturbed pointp is given by the cross product:**

  $\mathbf{n}' = \mathbf{p}'_u \times \mathbf{p}'_v$

  where

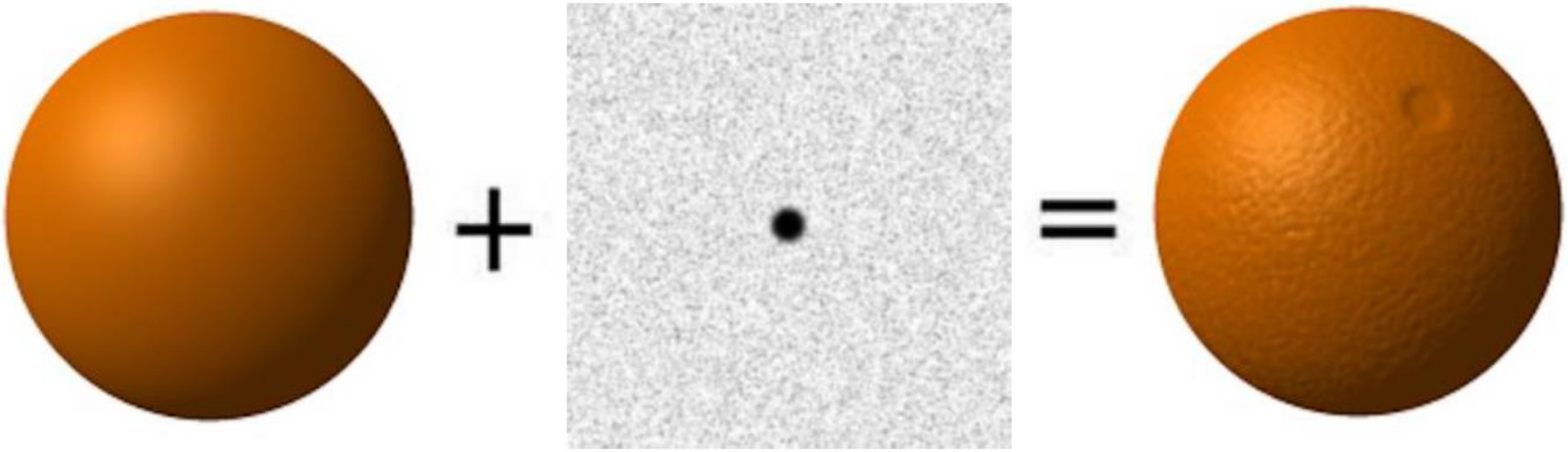
$$\mathbf{p}'_u = \mathbf{p}_u + \frac{\partial d}{\partial u}\mathbf{n} + d(u, v)\mathbf{n}_u$$

$$\mathbf{p}'_v = \mathbf{p}_v + \frac{\partial d}{\partial v}\mathbf{n} + d(u, v)\mathbf{n}_v$$

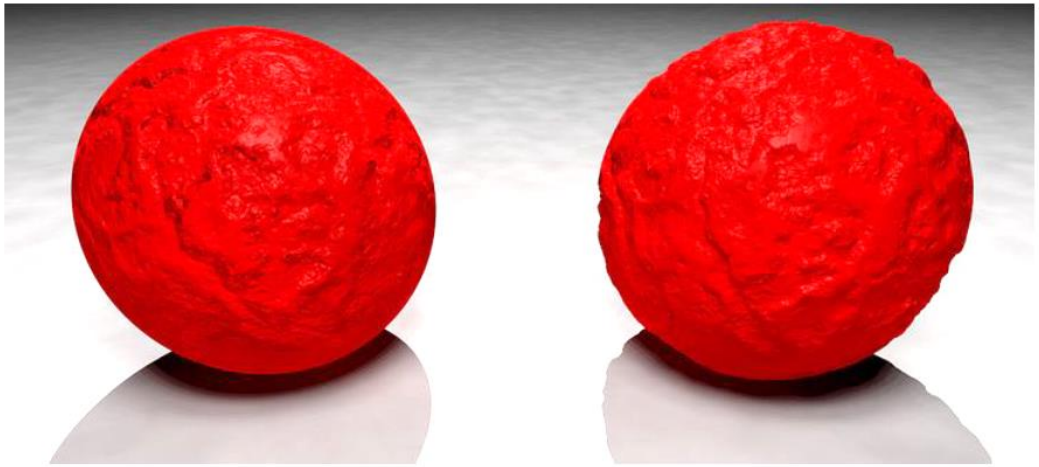
Method of perturbation(3)

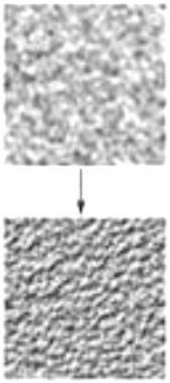
- **To obtain the approximate perturbed normal:**

$$\mathbf{n}' \approx \mathbf{n} + \frac{\partial d}{\partial u}\mathbf{n} \times \mathbf{p}_v + \frac{\partial d}{\partial v}\mathbf{n} \times \mathbf{p}_u$$

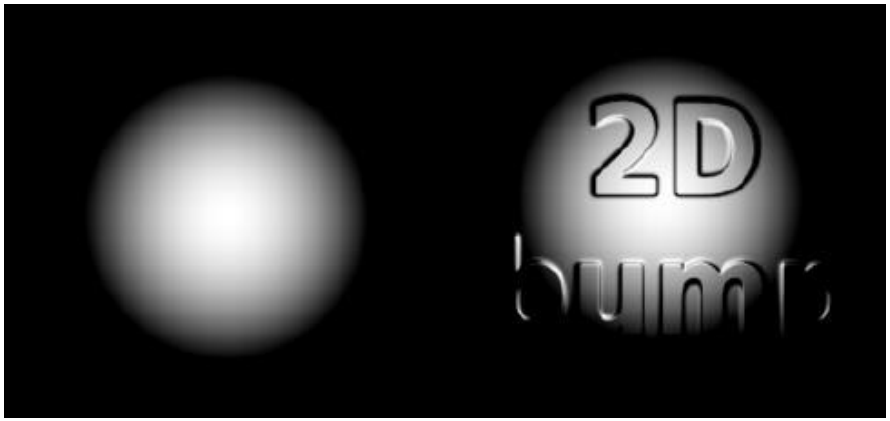


Example





Example



http://www.paulsprojects.net/tutorials/simplebump/simplebump.html

Displacement Mapping (置换贴图)

- Bump mapping adds realism, but it only changes the appearance of the object.
- We can do one better, and actually change the geometry of the object
  - This is displacement mapping
- Displacement mapping is an alternative computer graphics technique in contrast to bump mapping.
- Using a (procedural-) texture- or height map to cause an effect where **the actual** geometric position of points over the textured surface are **displaced**.

Displacement Mapping

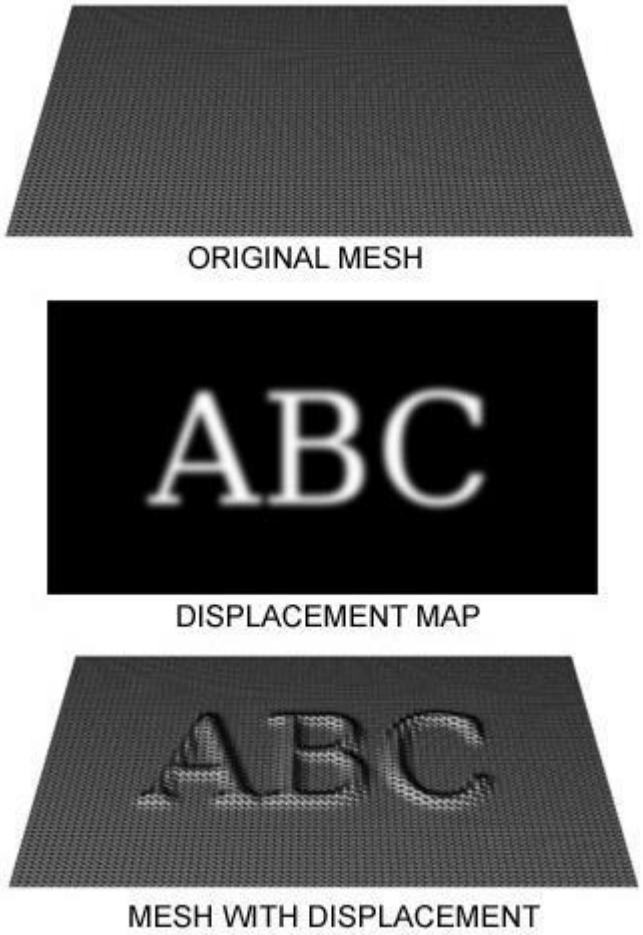
- Displacement mapping shifts all points on the surface in or out along their normal vectors
  - Assuming a displacement texture *d,*

    $\mathbf{p'} = \mathbf{p} + d(\mathbf{p}) * \mathbf{n}$

- Note that this actually changes the vertices, so it needs to happen in geometry processing

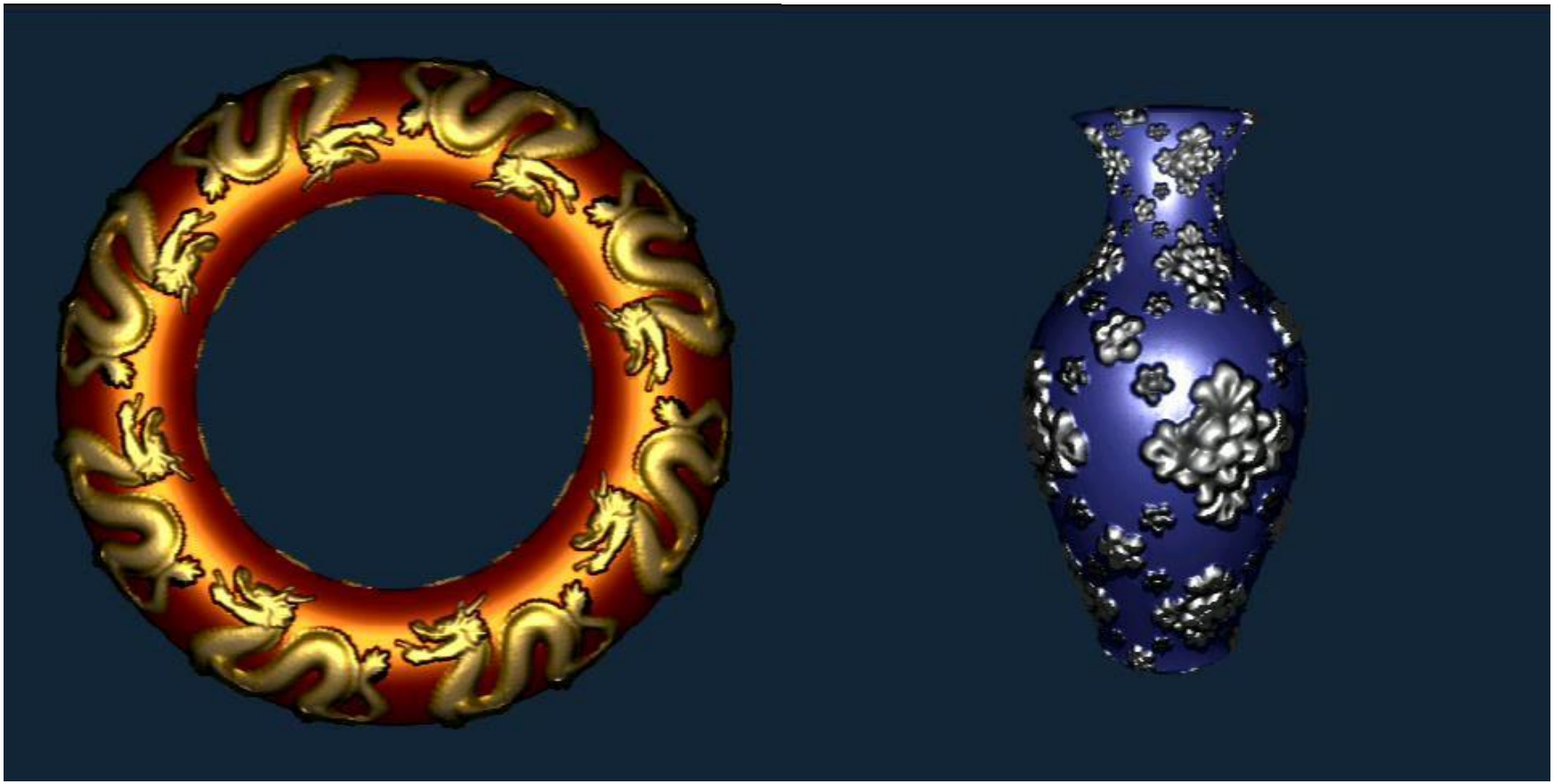
Displacement Mapping

- It gives surfaces a great sense of depth and detail, permitting in particular self-occlusion, self-shadowing and silhouettes.
- On the other hand, it is the most costly of this class of techniques owing to the large amount of additional geometry.



Example

View-Dependent Displacement Mapping



Displacement Mapping

Bump Mapping

Displacement Mapping

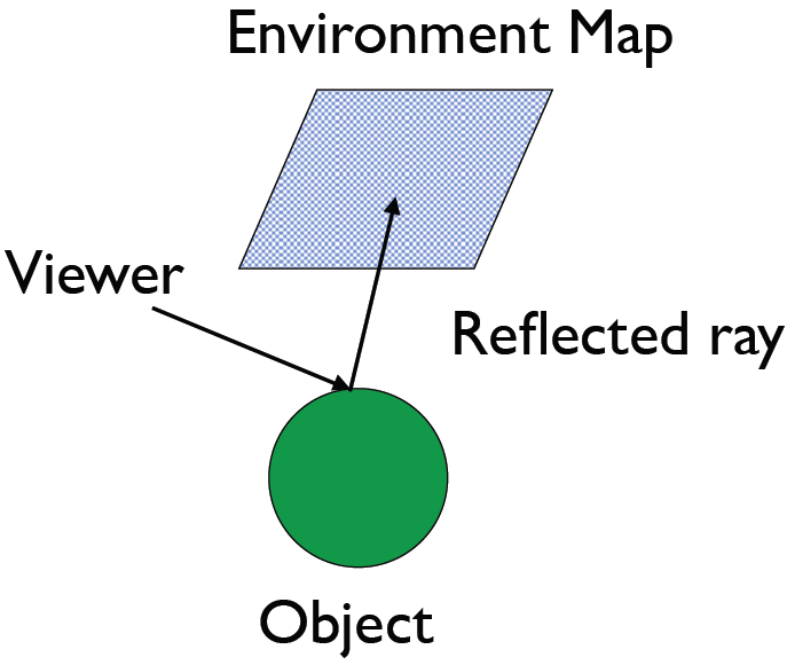
Environment Mapping

- Simulate complex mirror-like objects
  - Use textures to capture environment of objects
  - Use surface normal to compute texture coordinates

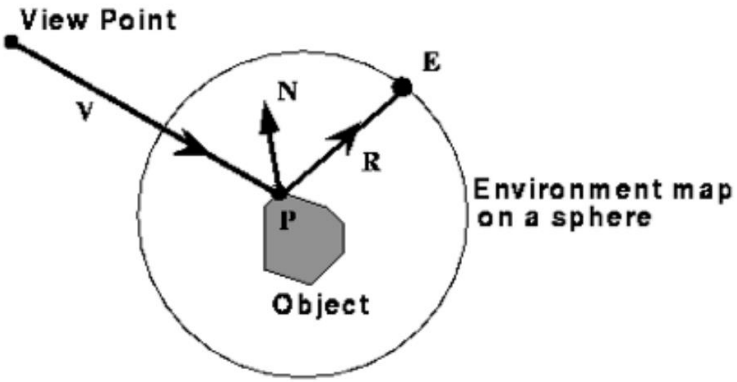
Environment Mapping

Environment Mapping

- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray:
  $R=2(N\cdot V)N-V$



Environment Mapping



- We use the *direction* of the reflected ray to index a texture map.
- We can simulate reflections. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.

Environment Mapping

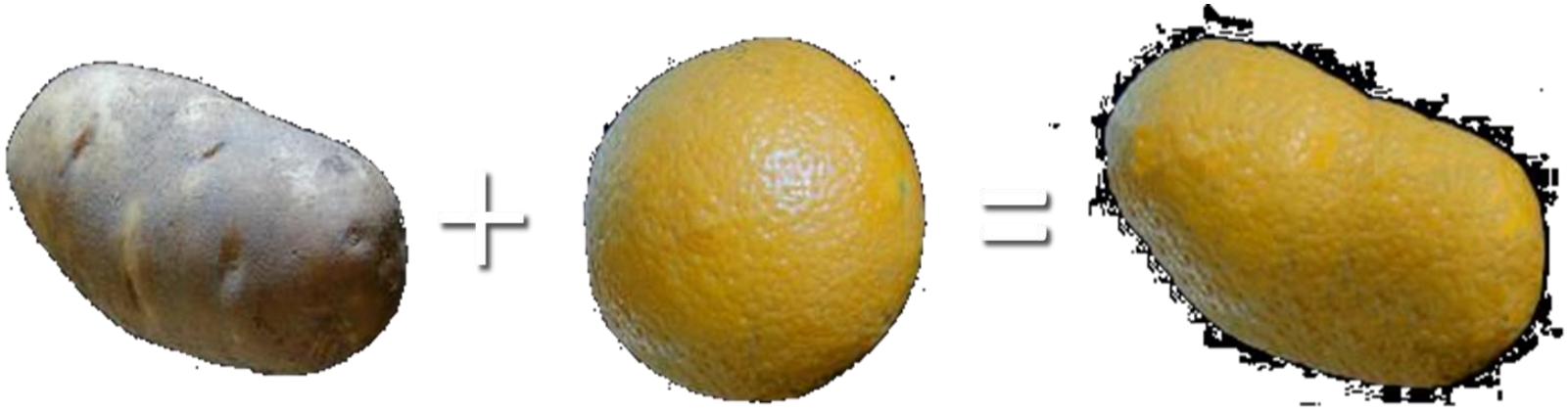
- A physically based rendering method, such as a ray tracer, can produce this kind of image.
- Ray-tracing calculations are too time-consuming to be practical for real-time applications
- Expand the texture mapping method：can give approximate results that are visually acceptable through environment maps.

Texture Application: Synthesis

Texture Application: Synthesis

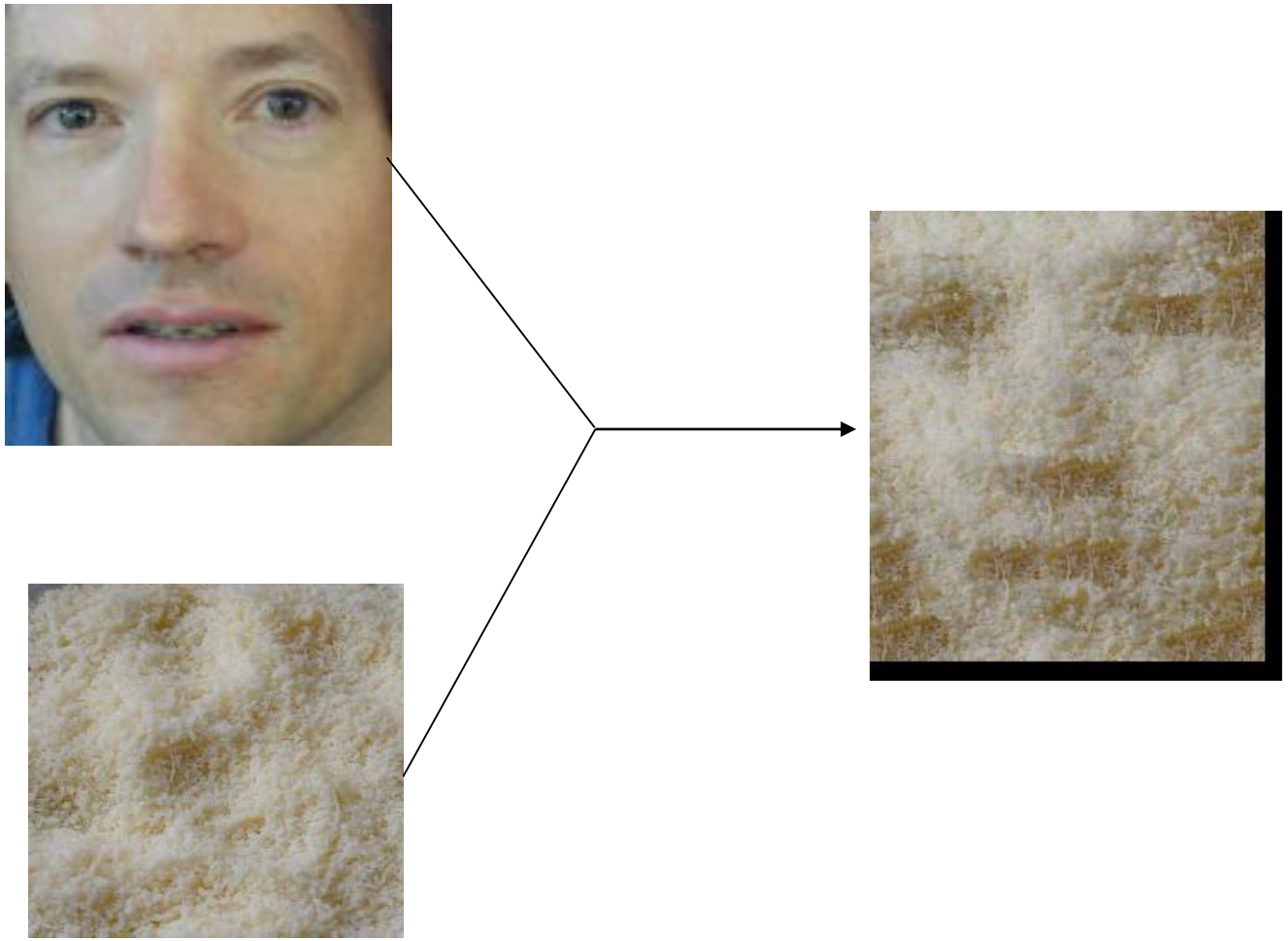
Texture Application: Texture Transfer

- Try to explain one object with bits and pieces of another object:



Texture Application: Texture Transfer

- Try to explain one object with bits and pieces of another object:



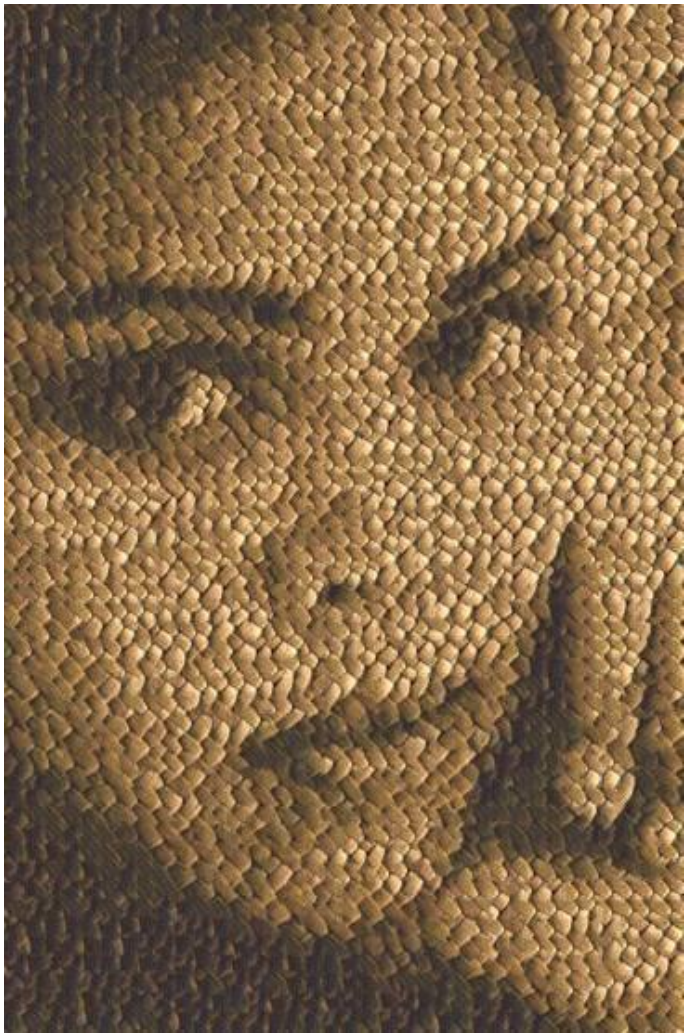
Texture Application: Texture Transfer

- Take the texture from one image and “paint” it onto another object

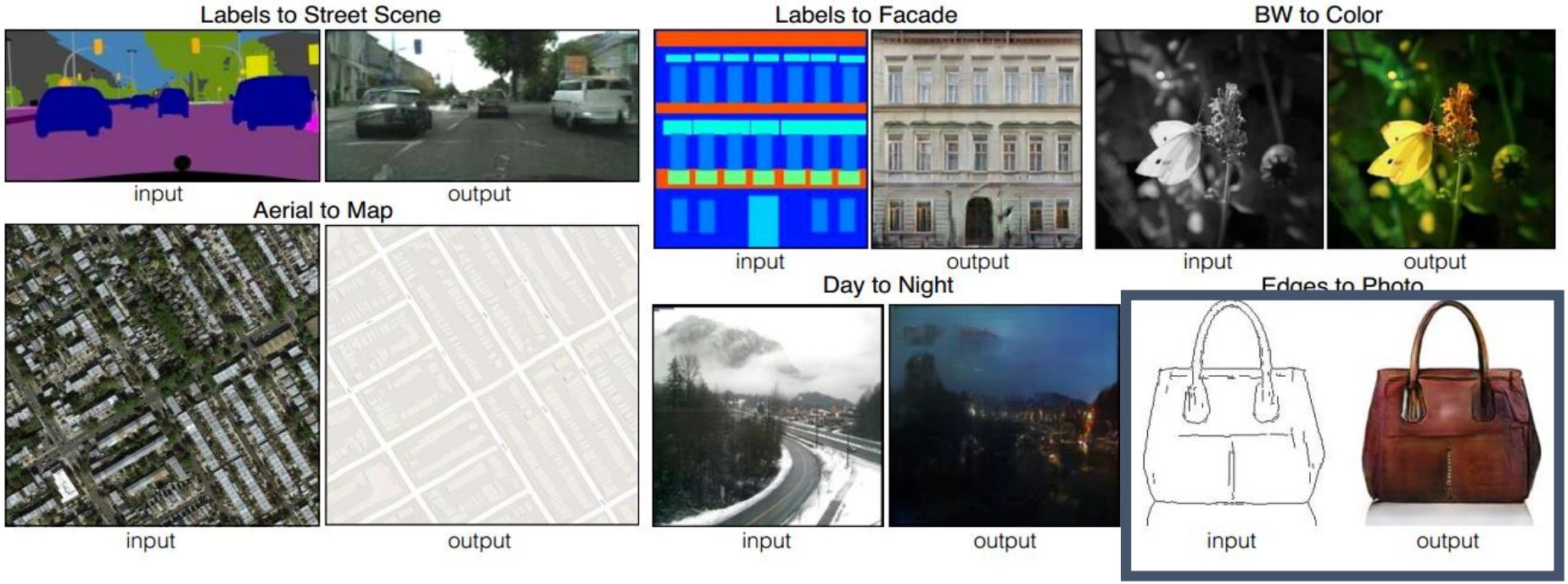
Same as texture synthesis, except an additional constraint:

1. Consistency of texture
2. Similarity to the image being “explained”

Texture Application: Texture Transfer



pix2pix: Image-to-Image Translation



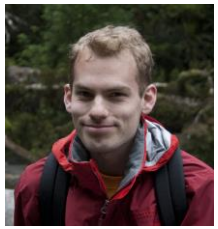
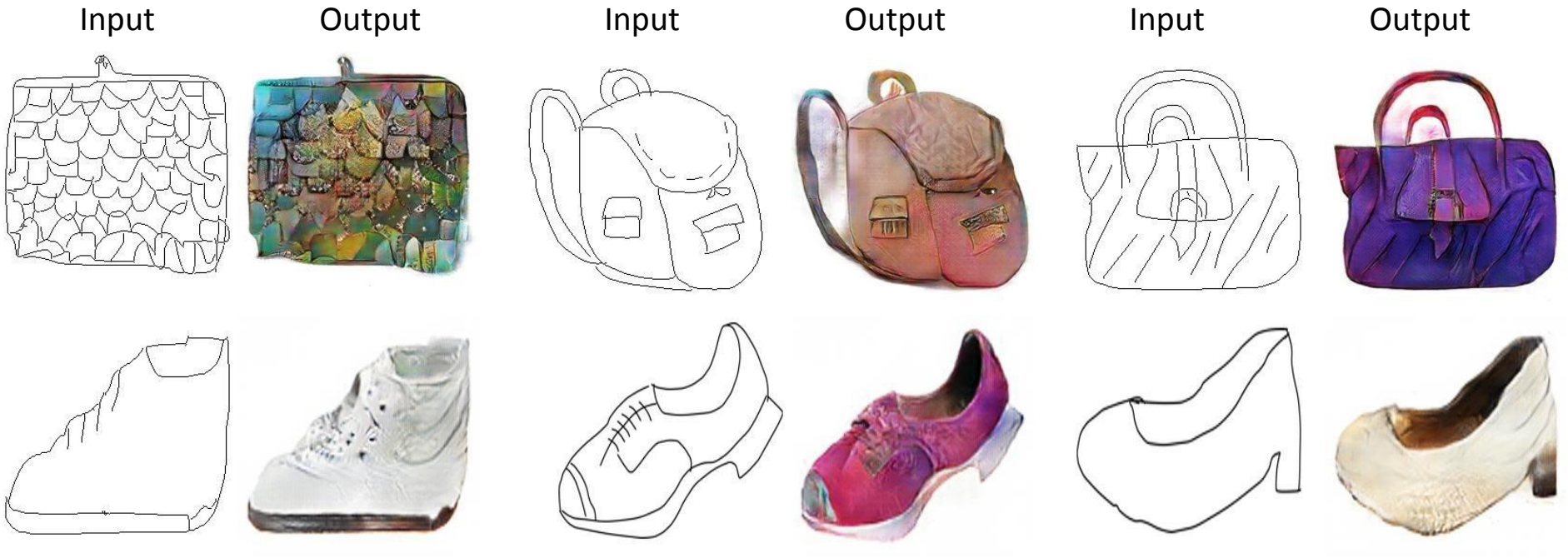


Image-to-image Translation with Conditional Adversarial Nets
Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. CVPR 2017

Sketches → Images



Trained on Edges → Images

Data from [Eitz, Hays, Alexa, 2012]

Collection Style Transfer

Photograph
@ Alexei Efros

Ukiyo-e Cezanne

Van Gogh Monet

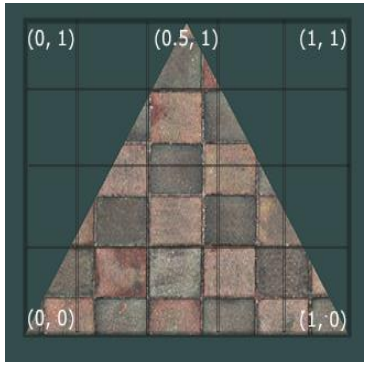
Collection Style Transfer

Texture Mapping In OpenGL

Texture

- Texture mapping（纹理图像的坐标与图形的空间坐标对应起来）
- You need to define texture coordinate points for the triangle's vertexes:
- 我们只需定义顶点对应的纹理坐标，图形内部对应的纹理图像的坐标是通过比例的方式计算出来的

纹理图像



三角形对应到纹理图像上

Specify Texture

- void glTexImage2D(target, level, internalformat, w, h, border, format, type, *texels);
  - ✓target: type of texture, e.g. GL_TEXTURE_2D
  - ✓ level: used for mipmapping, e.g. 0 (for no mip maps)
  - ✓ internalformat: elements per texel(纹理数据在OpenGL中是如何表示的), e.g. GL_RGB
  - ✓ w, h: width and height of texels in pixels, e.g. 256 * 256
  - ✓ border: used for smoothing , e.g. b = 0
  - ✓ format and type: describe texels(载入纹理的格式), e.g. GL_RGB
  - ✓ texels: pointer to texel array(actual texture image)

- If you have highest resolution image, OpenGL can generate low level prefiltered images automatically.
  - ✓ gluBuild2DMipmaps()
  - ✓gluBuild2DMipmapsLevels()
    - –To build only a subset

OpenGL filtering

- We can specify the kind of filter to be applied
- OpenGL filtering is crude but fast
- Specify filter using glParameteri() with
  - GL_TEXTURE_MAG_FILTER and GL_TEXTURE_MIN_FILTER
- filters :
  - GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_LINEAR (for minification)……
  - glParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);

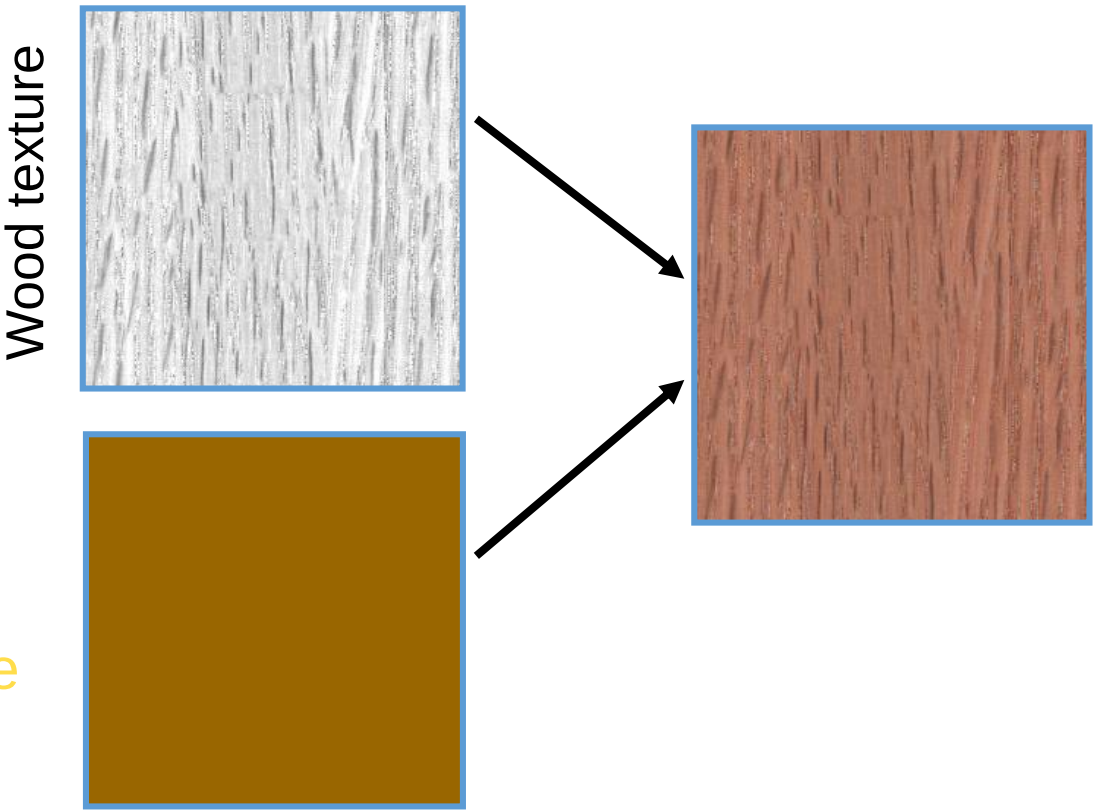
Texture filtering

- Instead of replacing color, you can also modulate color or some other property of polygon.
- glTexEnv(GL_TEXTURE_ENV, pname, param);
  - Pname and param specfiy how texture affects surface
- E.g.

glTexEnv(GL_TEXTURE_ENV,GL_TEXTURE_ENV_MODE,GL_MODULATE)

Modulation textures

Map texture values to scale factor

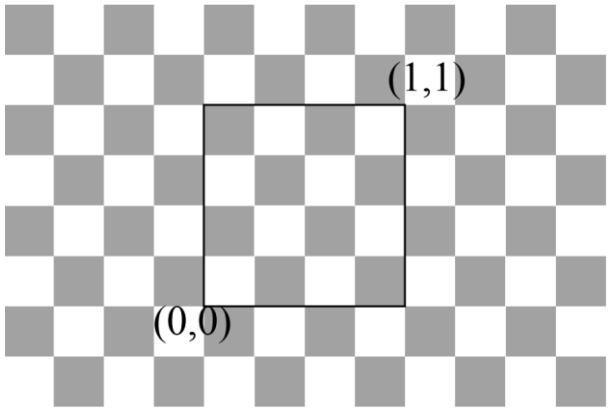


$$I = T(s,t)(I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L + K_T I_T + K_S I_S)$$

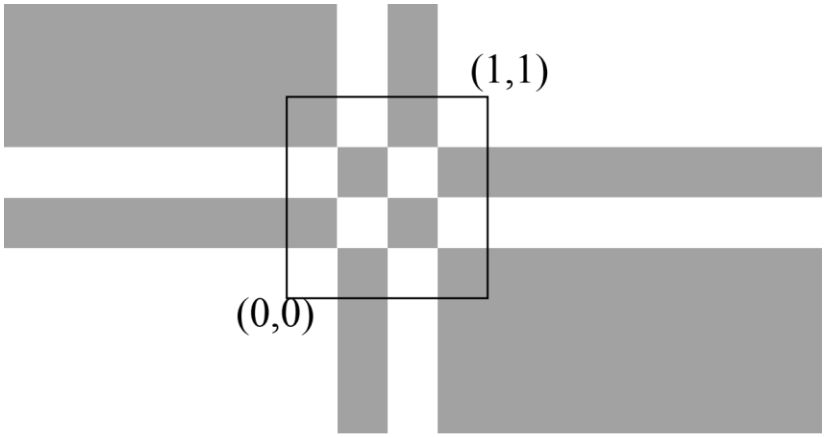
Texture Wrapping

- You can control what happens if a point maps to a texture coordinate outside of the texture image
  - All textures are assumed to go from (0,0) to (1,1) in texture space
- **Repeat**: Assume the texture is tiled
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
- **Clamp:** Clamp to Edge: the texture coordinates are truncated to valid values, and then used
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
- Can specify a special border color:
  - glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, R,G,B,A)

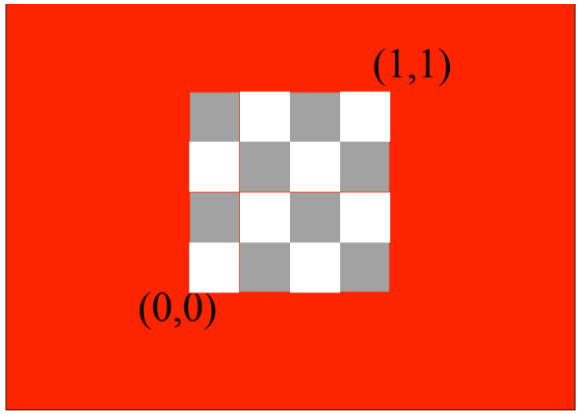
Texture Wrapping



Repeat Border



Clamp Border



Border Color

Texture Wrapping

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

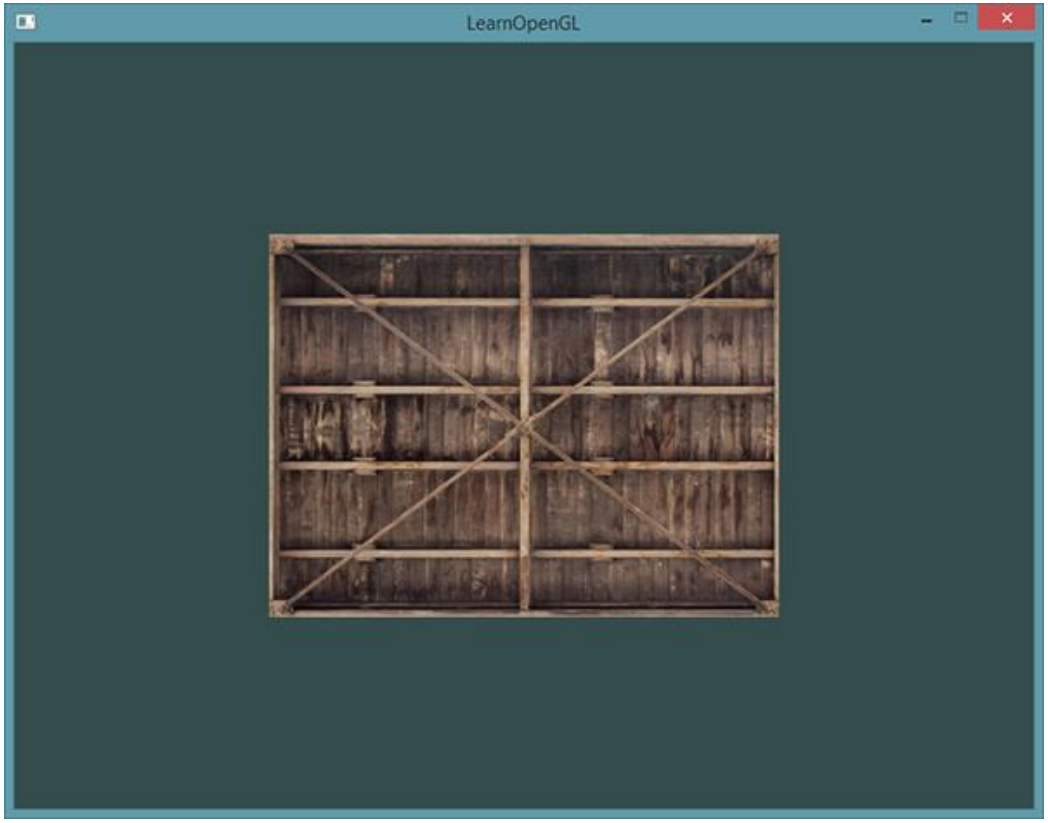
GL_CLAMP_TO_BORDER

```
float borderColor[] = { 1.0f, 1.0f, 0.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
```

这是定义GL_CLAMP_TO_BORDER的方式

Texture Mapping in OpenGL

- Goal：Apply texture to a rectangle



Texture Mapping in OpenGL

- Goal：Apply texture to a rectangle

1. Prepare texture data
2. Prepare model data
3. Draw the model with texture

Texture Mapping in OpenGL

1.Prepare the texture data

- Load image file
- Create a texture object
- Bind the texture object to the current texture object reference
- Attach image data to currently bound texture object

Then you can use the texture data in vertex shader and fragment shader

1.Prepare the texture data

- Load texture image file
  - use **stb_image.h** or other ways

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

```
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);
```

纹理图像：container.jpg

1.Prepare the texture data

- Create a texture object

```
unsigned int texture;
glGenTextures(1, &texture);
```

- Bind the texture object to the current texture object reference

```
glBindTexture(GL_TEXTURE_2D, texture);
```

- Attach image data to currently bound texture object

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glGenerateMipmap(GL_TEXTURE_2D);
```

1.Prepare the texture data

- The whole process of generating a texture

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
// set the texture wrapping/filtering options (on the currently bound texture object)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
// load and generate the texture
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);
if (data)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
}
else
{
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(data);
```

2. Prepare model data

- load from model file or specify it by code
  - Specify rectangle information
    - 3D position of vertexes
    - color of vertexes
    - texture coordinates of vertexes

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,   // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,   // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f    // top left
};
```

矩形模型数据

- We usually draw with triangle, so we should specify the triangle's vertex(rectangle vertex data's indexes)

```
unsigned int indices[] = {
    0, 1, 3, // first triangle
    1, 2, 3  // second triangle
};
```

绘制的矩形的两个三角形的顶点索引

2. Prepare model data

- Like Texture Object, We use VAO, VBO, EBO to store model data
- Program will read model data from them when drawing
  - Create VAO, VBO, EBO

```
unsigned int VBO, VAO, EBO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
glGenBuffers(1, &EBO);
```

  - Bind the VAO to current VAO

```
glBindVertexArray(VAO);
```

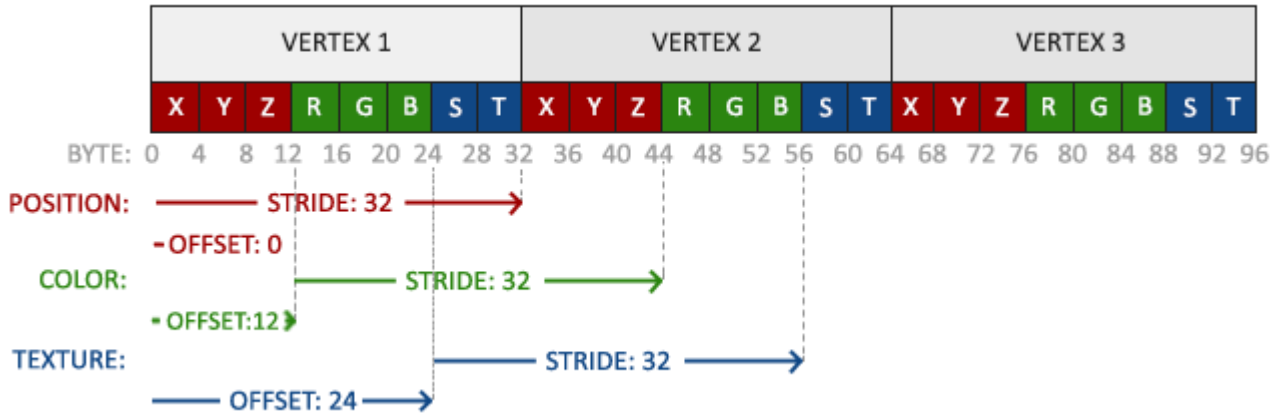
  - Bind the VBO and the EBO to currently bound VAO, and bind the model data to VBO and EBO

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);
```

2. Prepare model data

- Applying textures：Use glVertexAttribPointer function to tell how OpenGL should interpret the vertex buffer data when drawing call is made

```
// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// color attribute
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
// texture coord attribute
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```



2. Prepare model data

- Vertex shader（GPU Program）
  - get the data in vertex buffer
- Fragment shader（GPU Program）
  - map each pixel to texture

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main()
{
    gl_Position = vec4(aPos, 1.0);
    ourColor = aColor;
    TexCoord = aTexCoord;
}
```

vertex shader

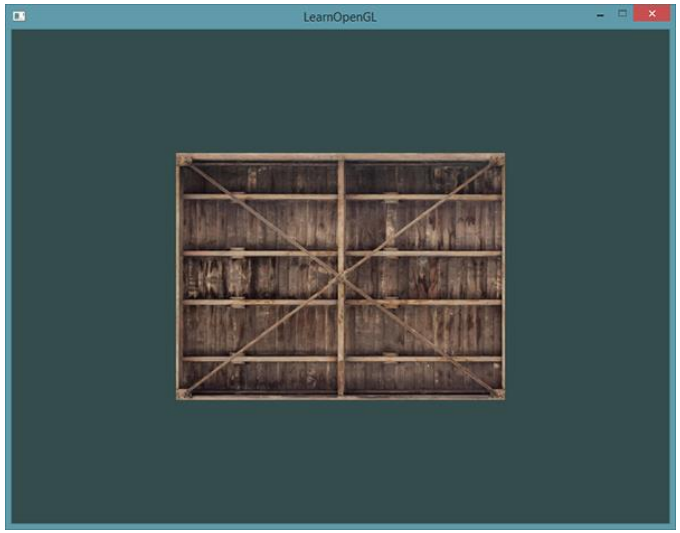
```
#version 330 core
out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

uniform sampler2D ourTexture;

void main()
{
    FragColor = texture(ourTexture, TexCoord);
}
```

fragment shader

3. Draw the model with texture

- at last, active the shader program, bind texture and vertex buffer object, and then draw it!

```
// render container
ourShader.use();
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```



3. Draw the model with texture

- we haven’t use the color of rectangle itself!
- we can also mix the resulting texture color with the vertex colors.
- In Fragment Shader:

```
FragColor = texture(ourTexture, TexCoord) * vec4(ourColor, 1.0);
```