Computer Graphics 2016

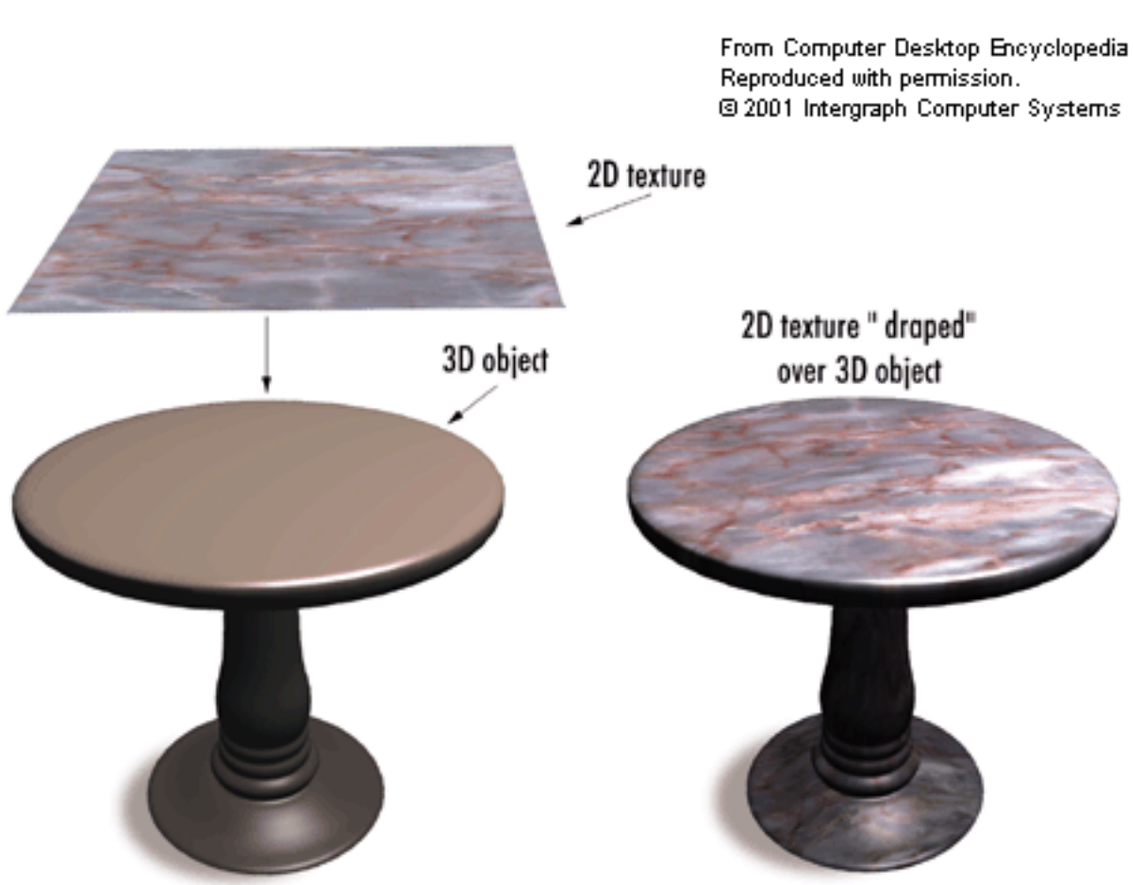
13. Texture Mapping

Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

2016-12-26

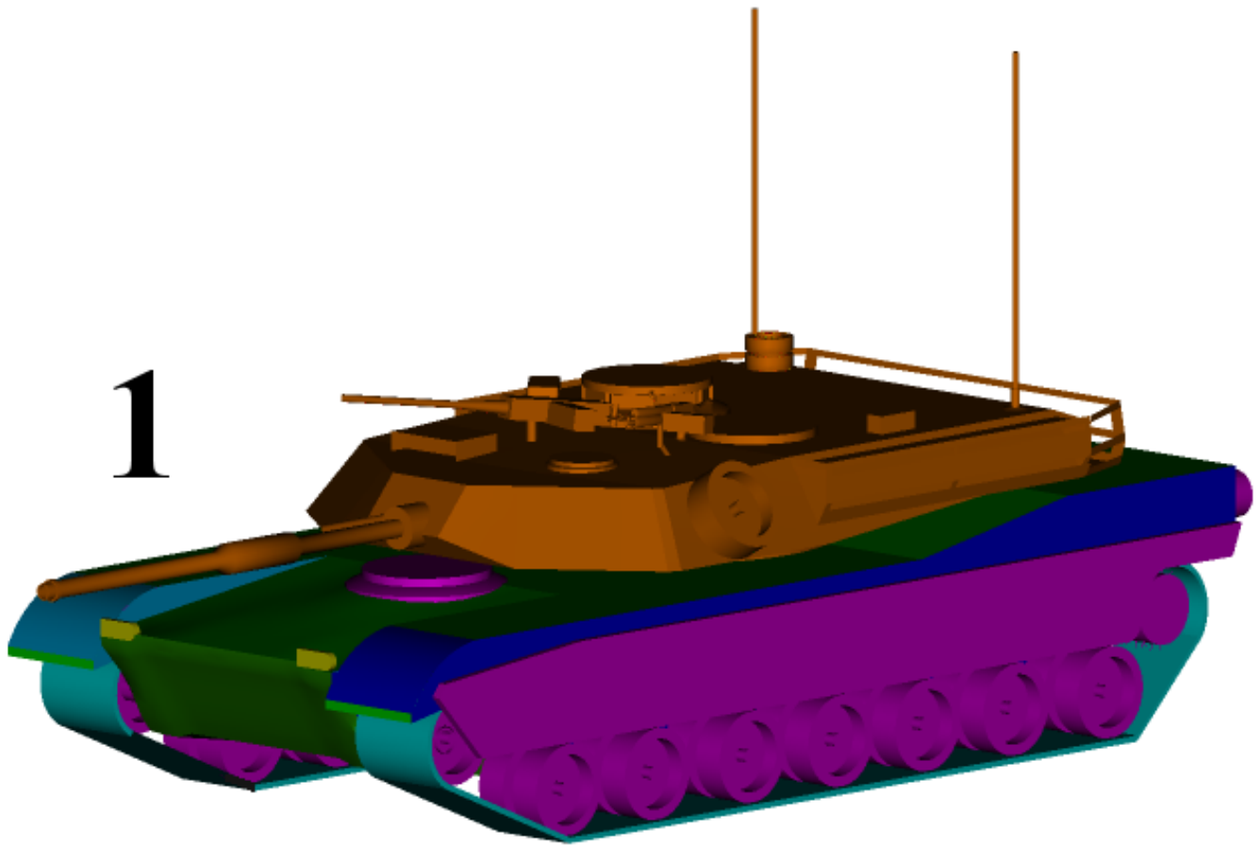
Texture Mapping

- So far, every object has been drawn either in a solid color, or smoothly shaded between the colors at its vertices.
  - **Similar to painting**
- Texture Mapping applies a variation to the surface properties of the object instead
  - **Similar to surface finishing**, example like wallpaper on a wall surface or sticking a label onto a bottle.

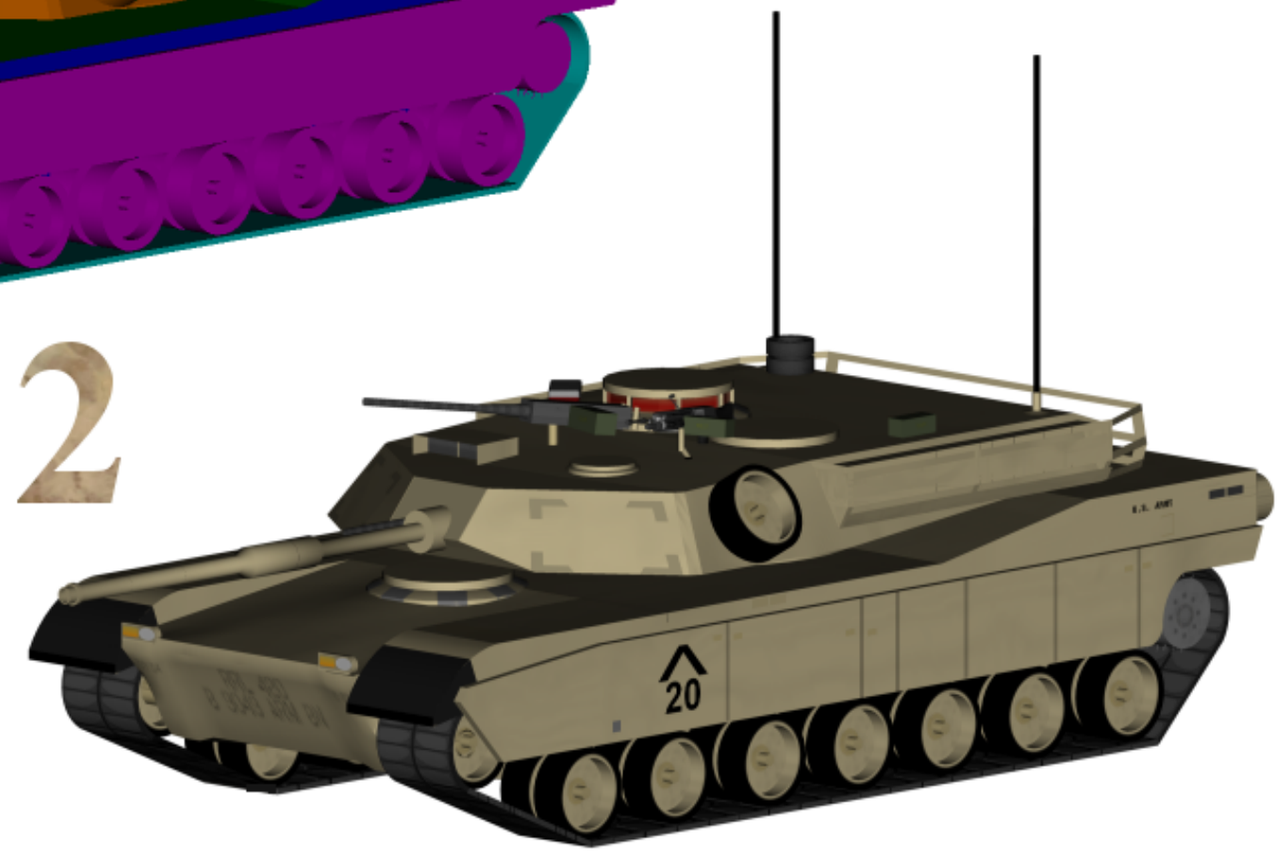


Texture Mapping

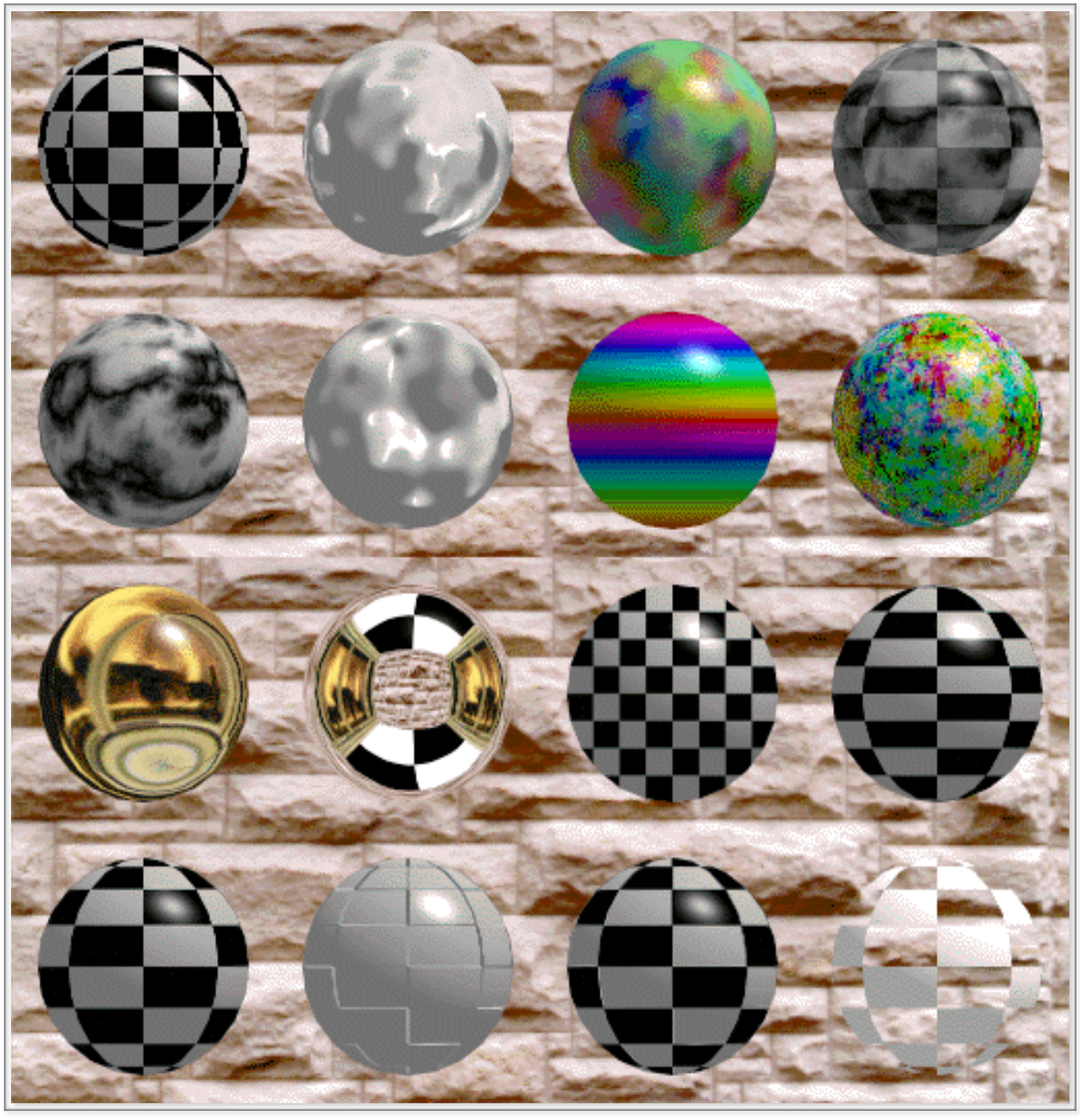
1



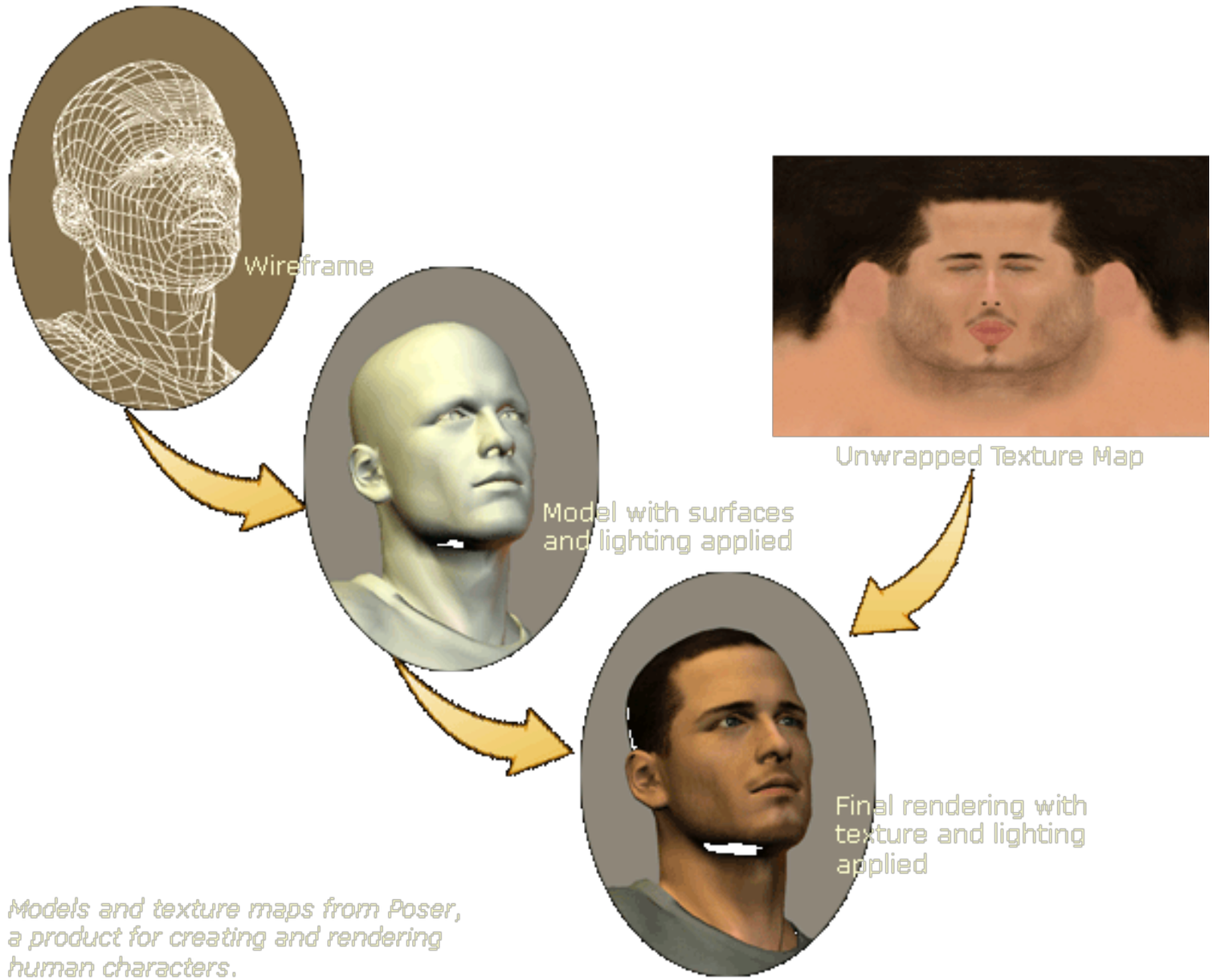
2



Texture Mapping



Texture Mapping



Texture Mapping

Texture mapping

AD: Genetica - a texture generation tool

Texture mapping

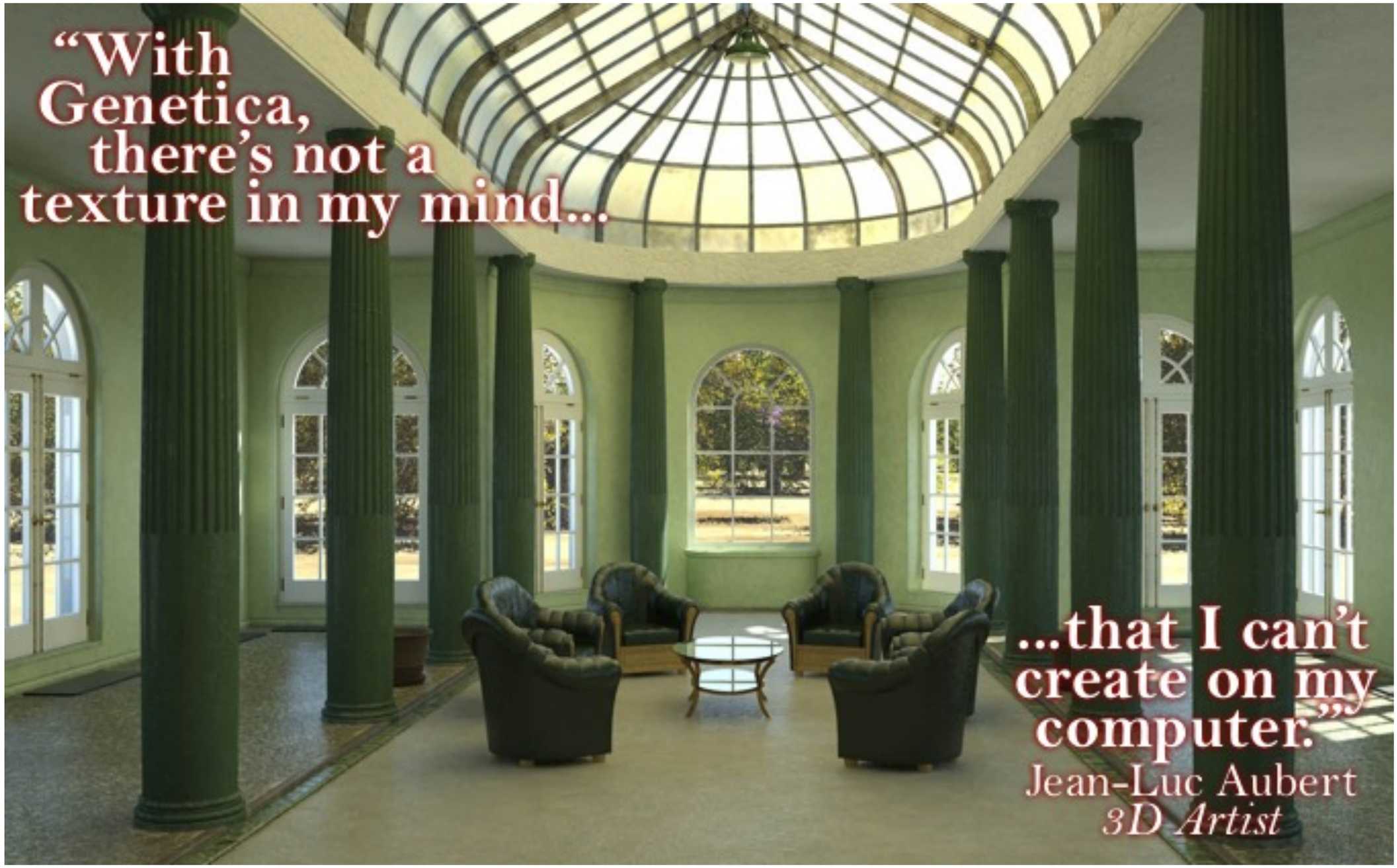
AD: Genetica - a texture generation tool

Texture mapping

AD: Genetica - a texture generation tool

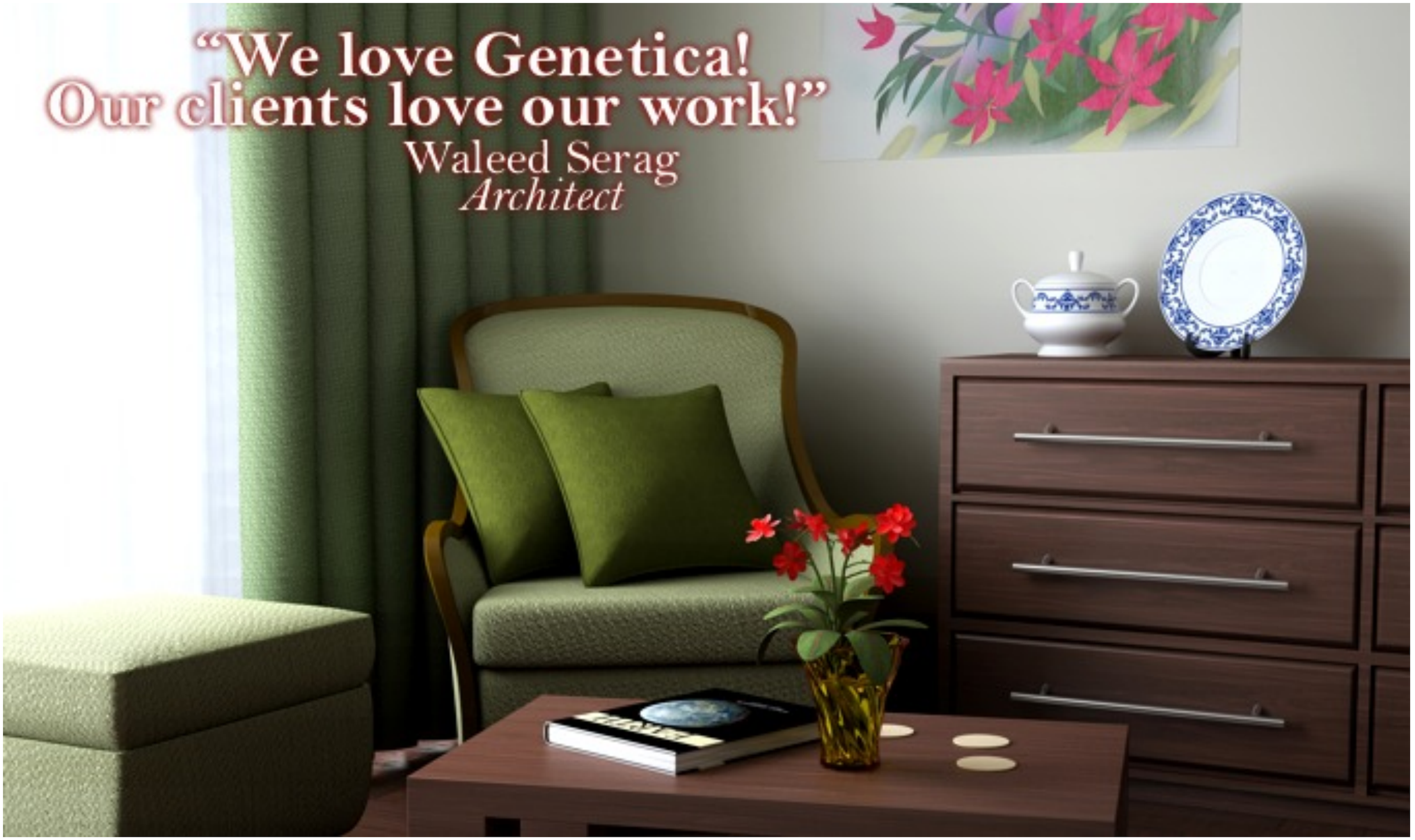
Texture mapping

AD: Genetica - a texture generation tool



Texture mapping

AD: Genetica - a texture generation tool



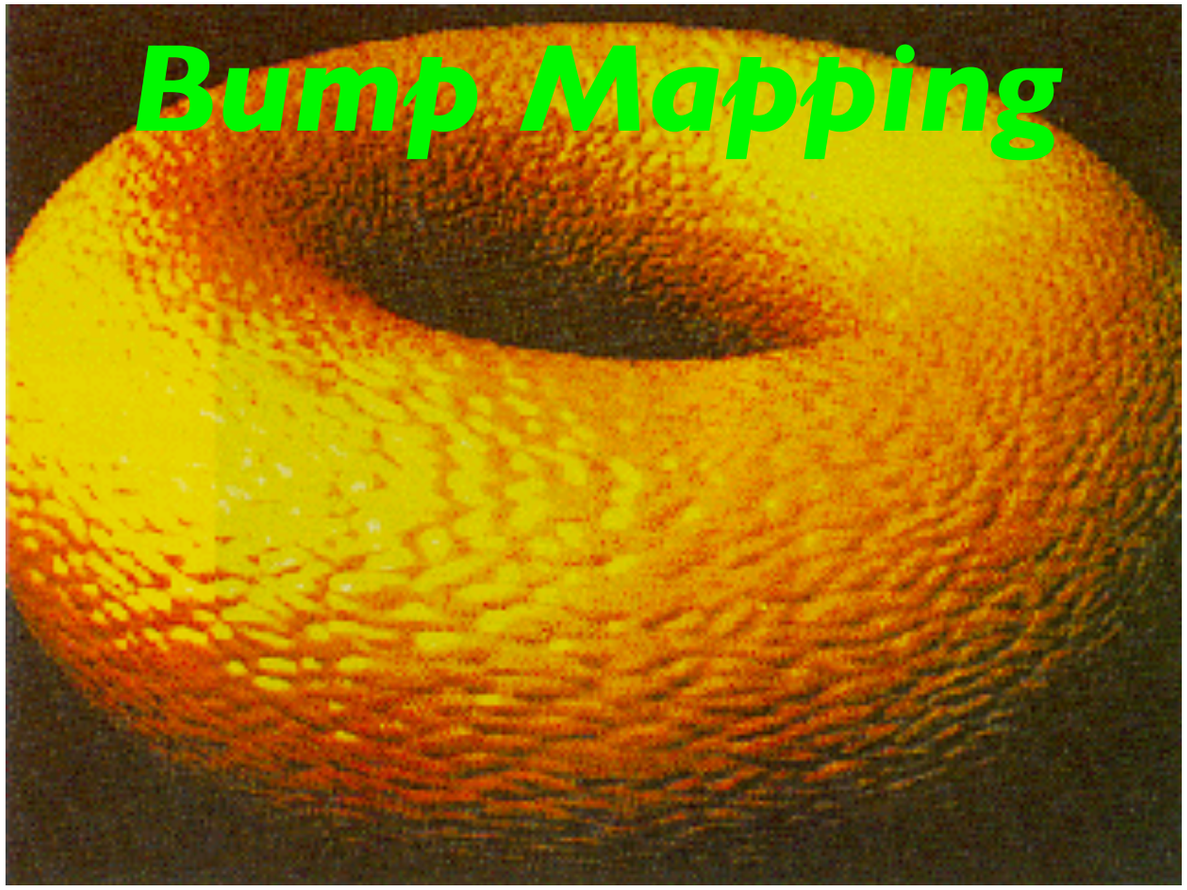
Why Texture mapping

- Texture is variation in the surface attributes
  - like color, surface normals, specularity, transparency, surface displacement etc.
- Computer generated images look more realistic if they are able to capture these complex details
- It is difficult to represent these details using geometric modeling because they heavily increase computational requirements
- Texture mapping is an effective method of "faking" surface details at a relatively low cost

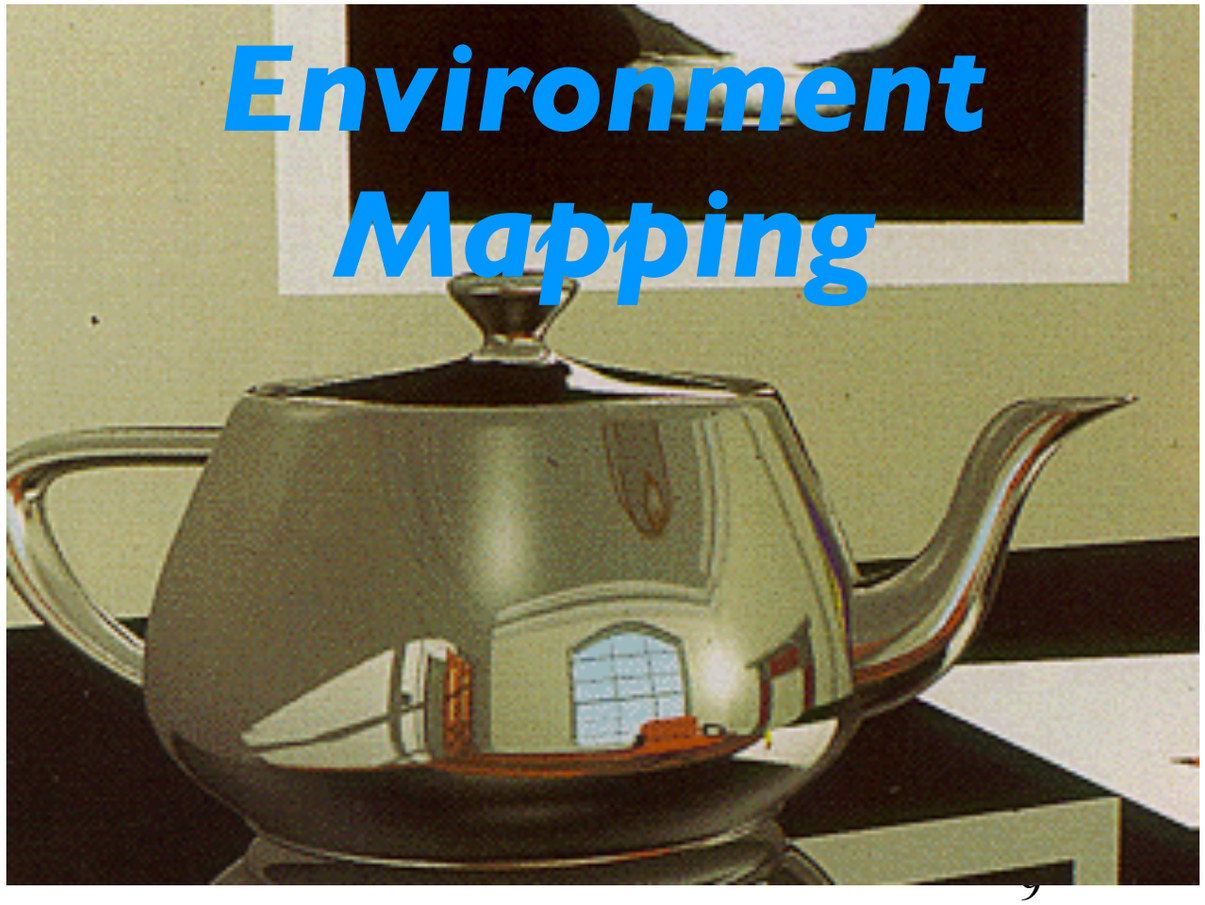
What is texture mapping

- Texture mapping is the process of transforming a texture on to a surface of a 3D object.
- 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.

Image Mapping



Bump Mapping



Environment Mapping

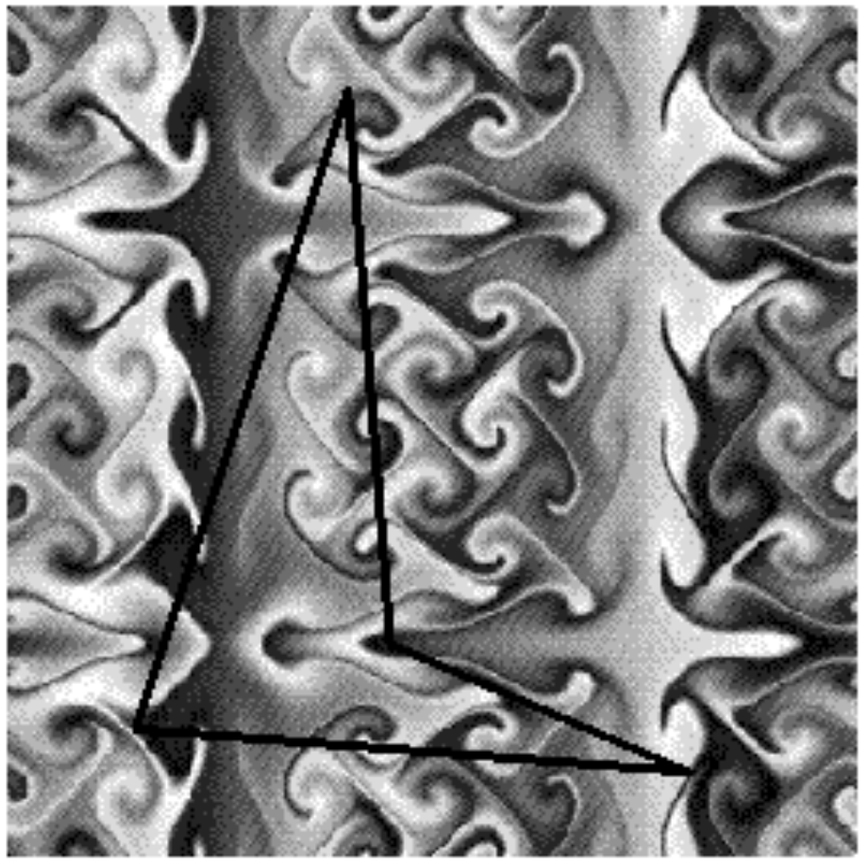
For what kind of objects?

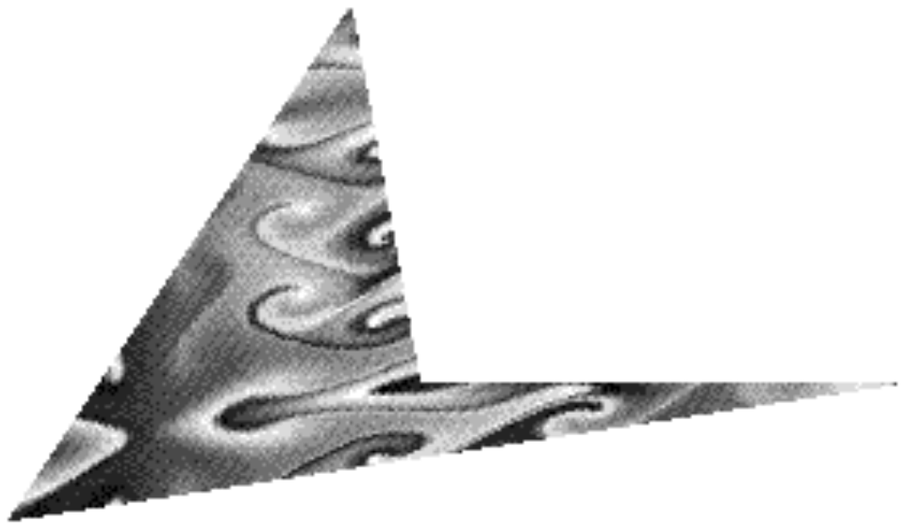
- In general Texture mapping for arbitrary 3D objects is difficult
  - E.g. Distortion (try mapping a planer texture onto sphere)
- It is easiest for polygons and parametric surfaces
- We limit our discussion to Texture mapping polygons

What is a typical 2D texture?

- A 2D function represented as rectangular array of data
  - Color data
  - Luminance data
  - Color and alpha data
- 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)

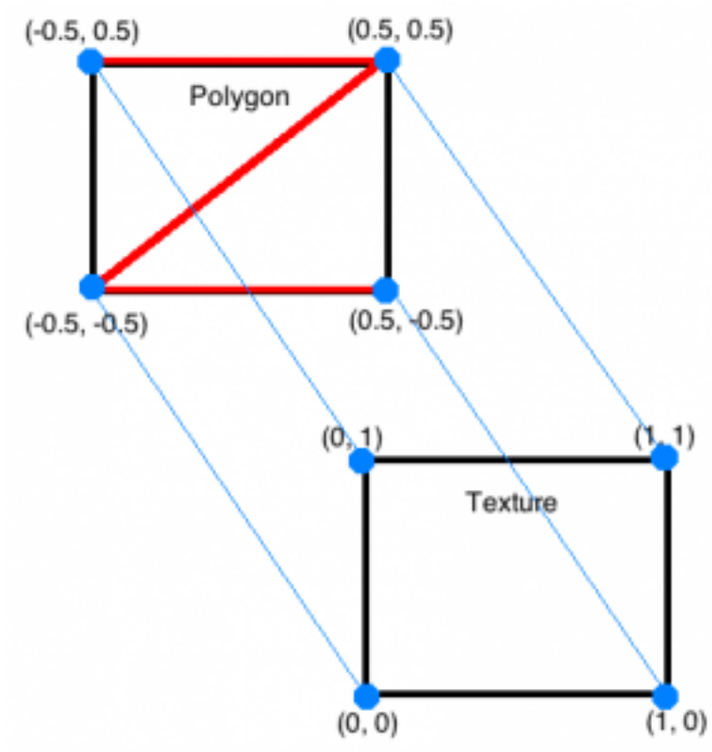
Example





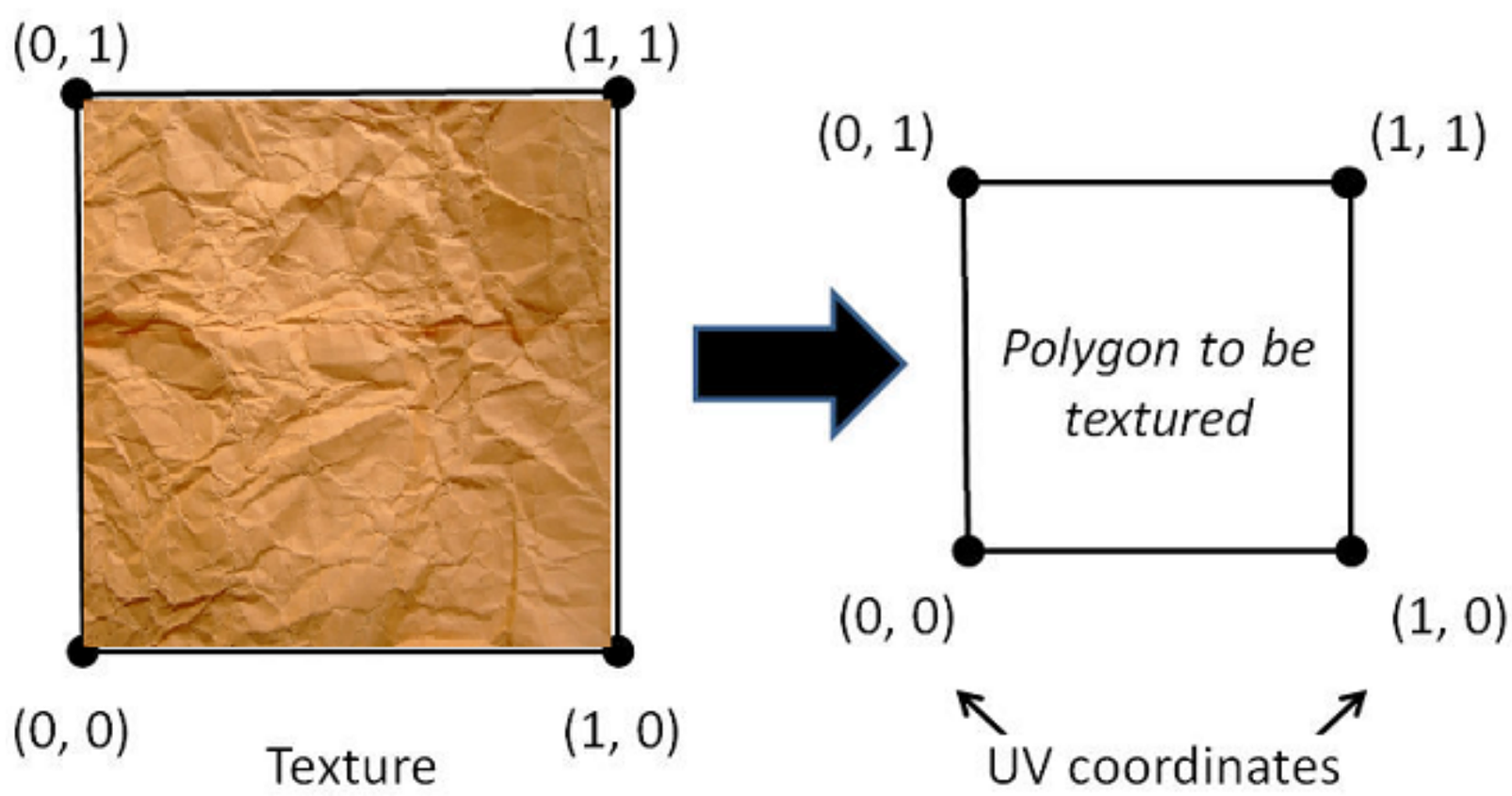
Assigning Texture Coordinates

- 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 *s* and *t* to distinguish them from the *x*, *y* and *z* coordinates of 3D space.
- A vertex's texture coordinates determine which texel(s) are mapped to the vertex.



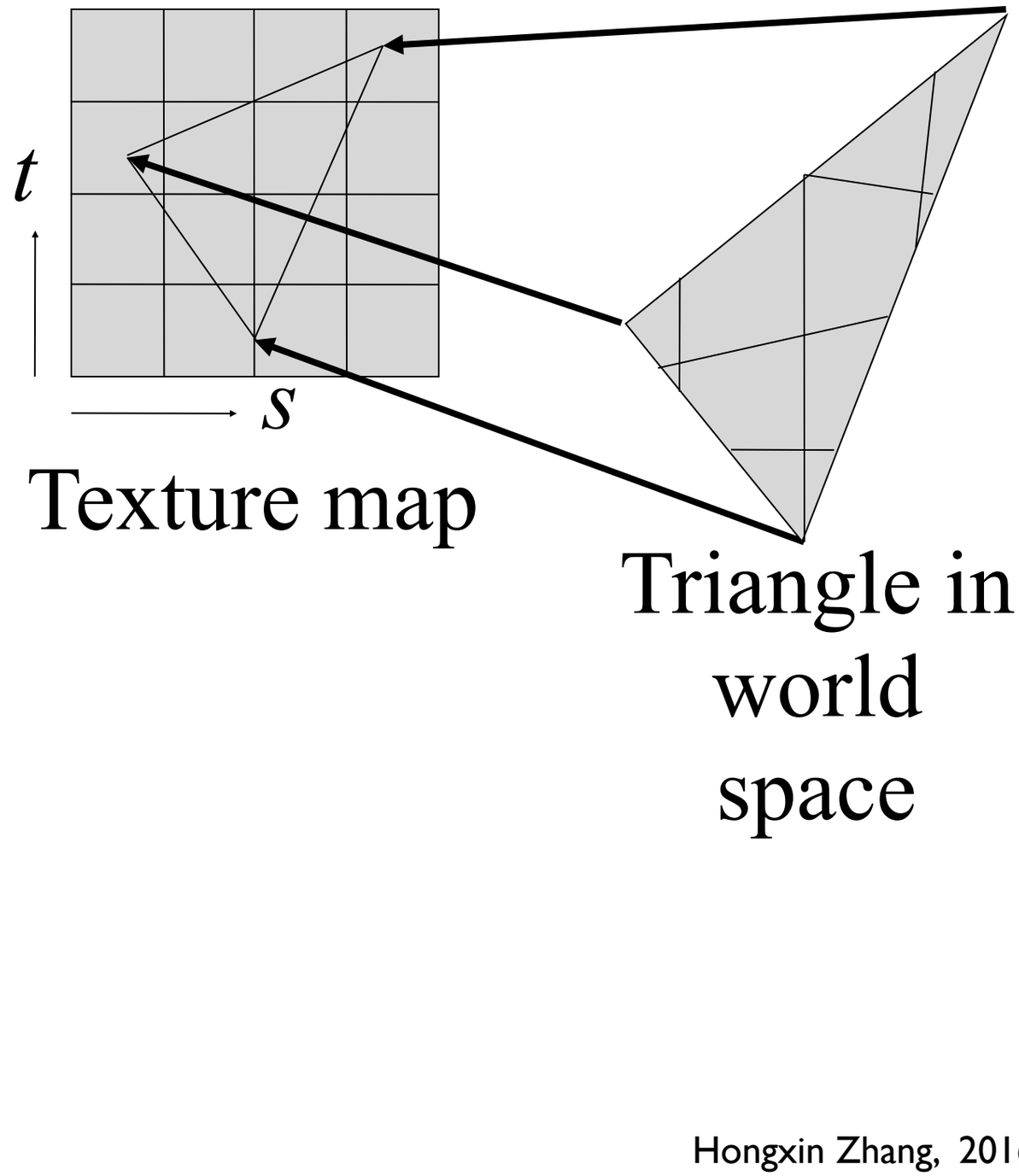
Assigning Texture Coordinates

- 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.

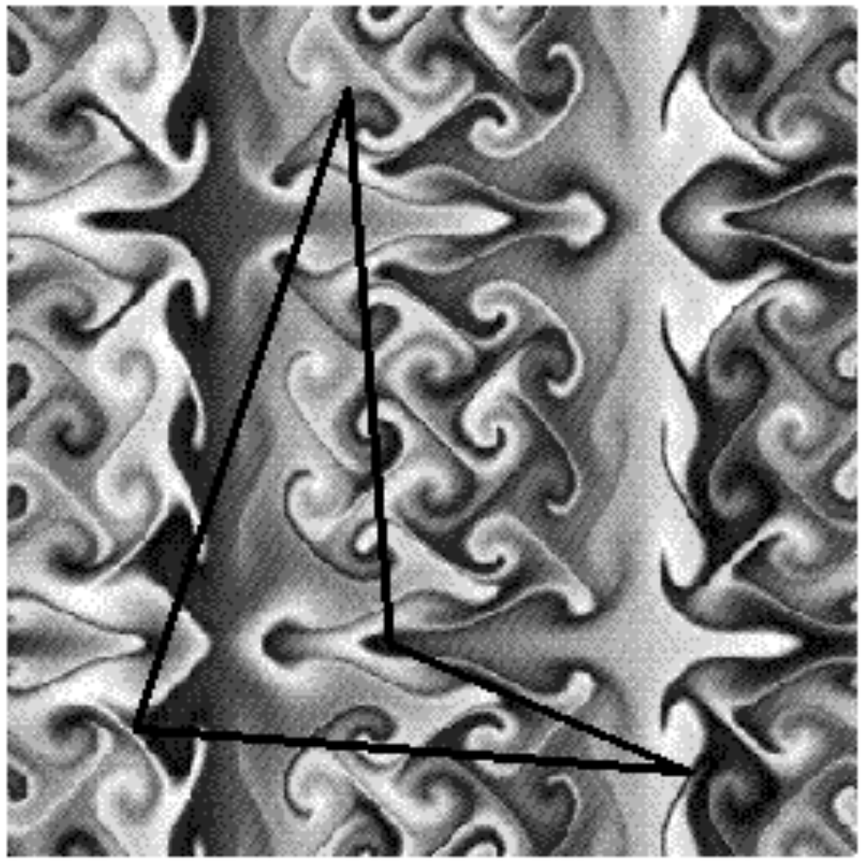


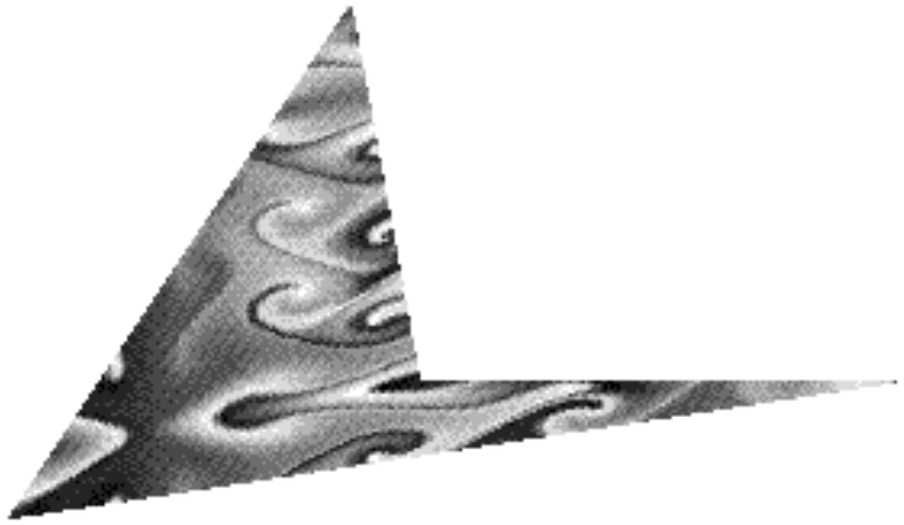
Texture Interpolation

- 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
  - But …



Texture Example



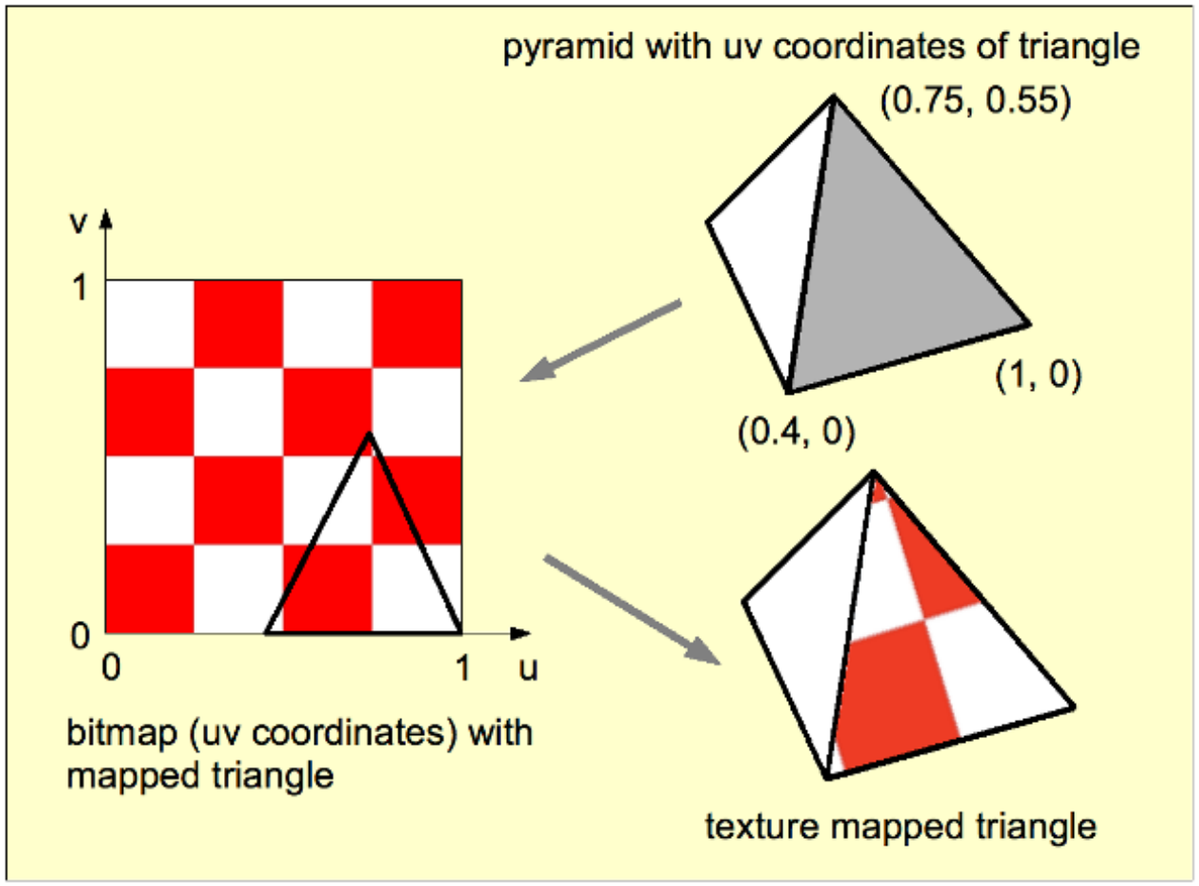


Polygonal texture mapping

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

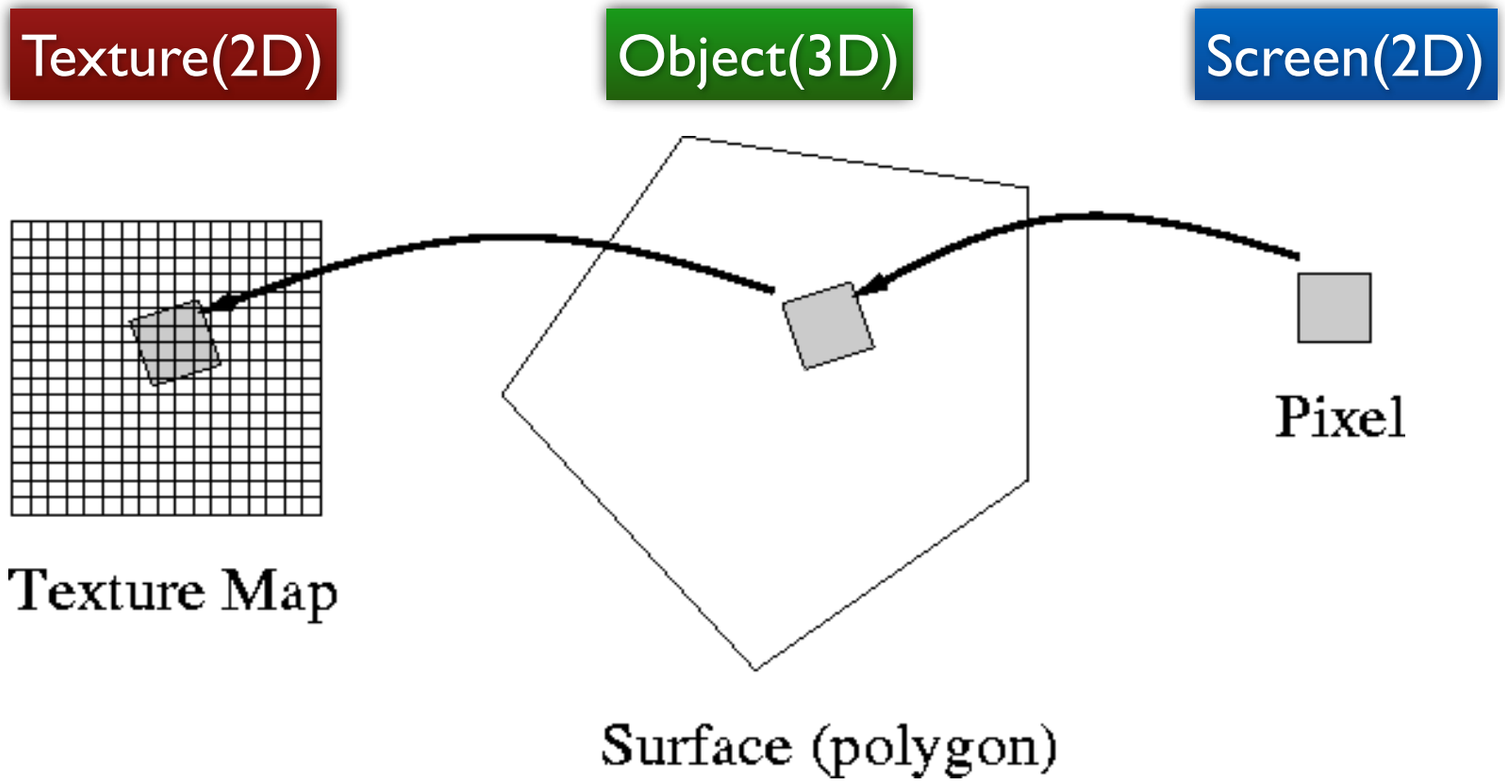
Establish Correspondences

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



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.

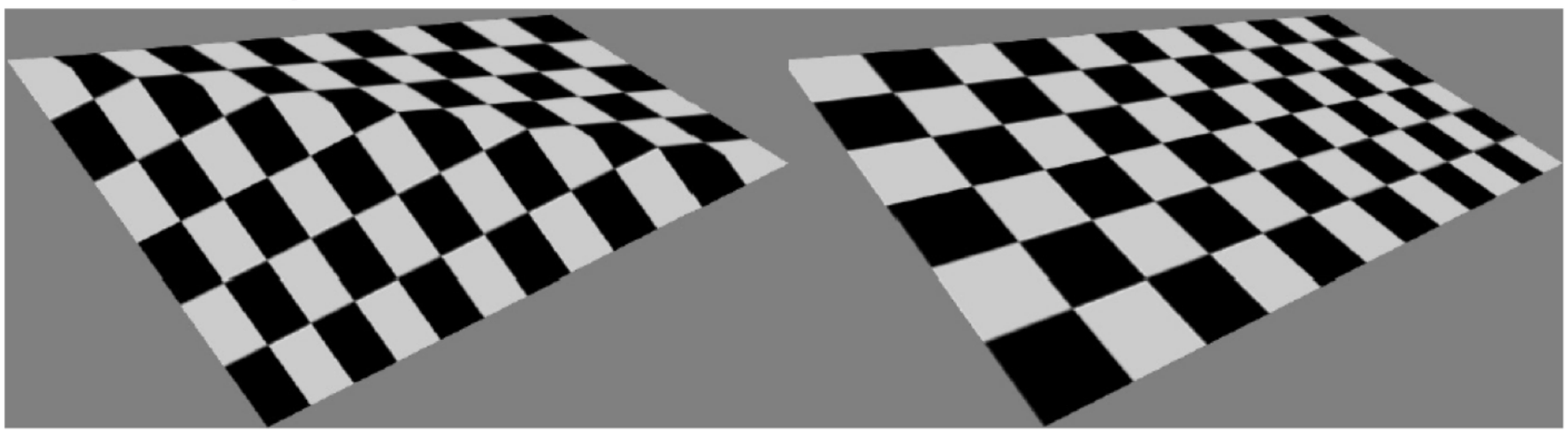


Crude texture mapping code

```
// 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;
}
```

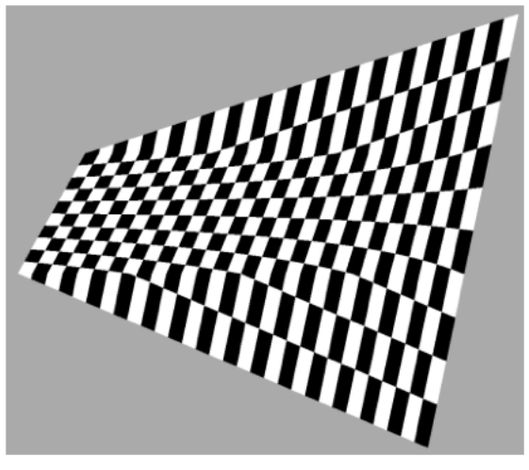
Artifacts

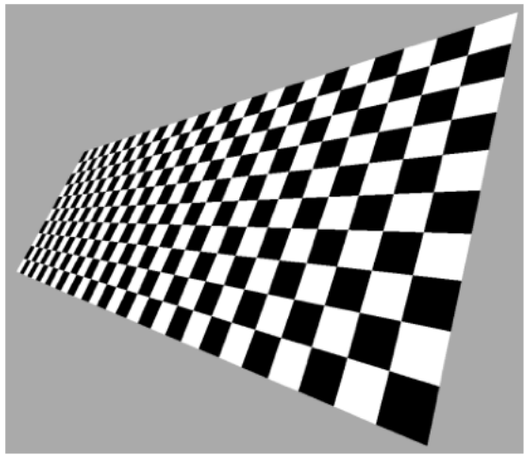
- two triangles in the same plane



- a quadrilateral

[Akenine-Moeller et al.: Real-time Rendering]



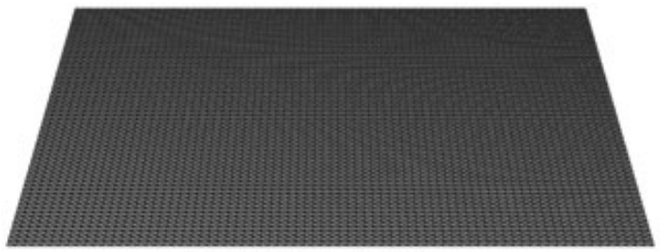


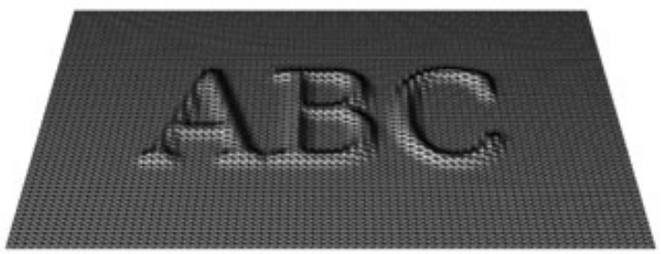
[Heckbert, Moreton]

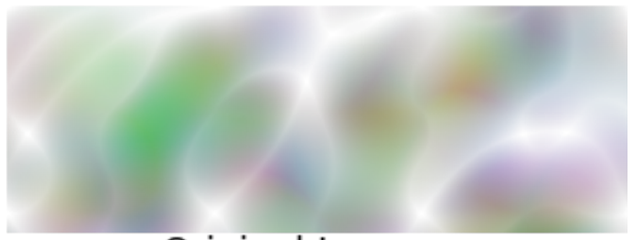
Such distortion may be reduced with subdivision

Extensions to code

- 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






Original Image

Displacement Map

Image with Displacement

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.

Attributes modulated for texture

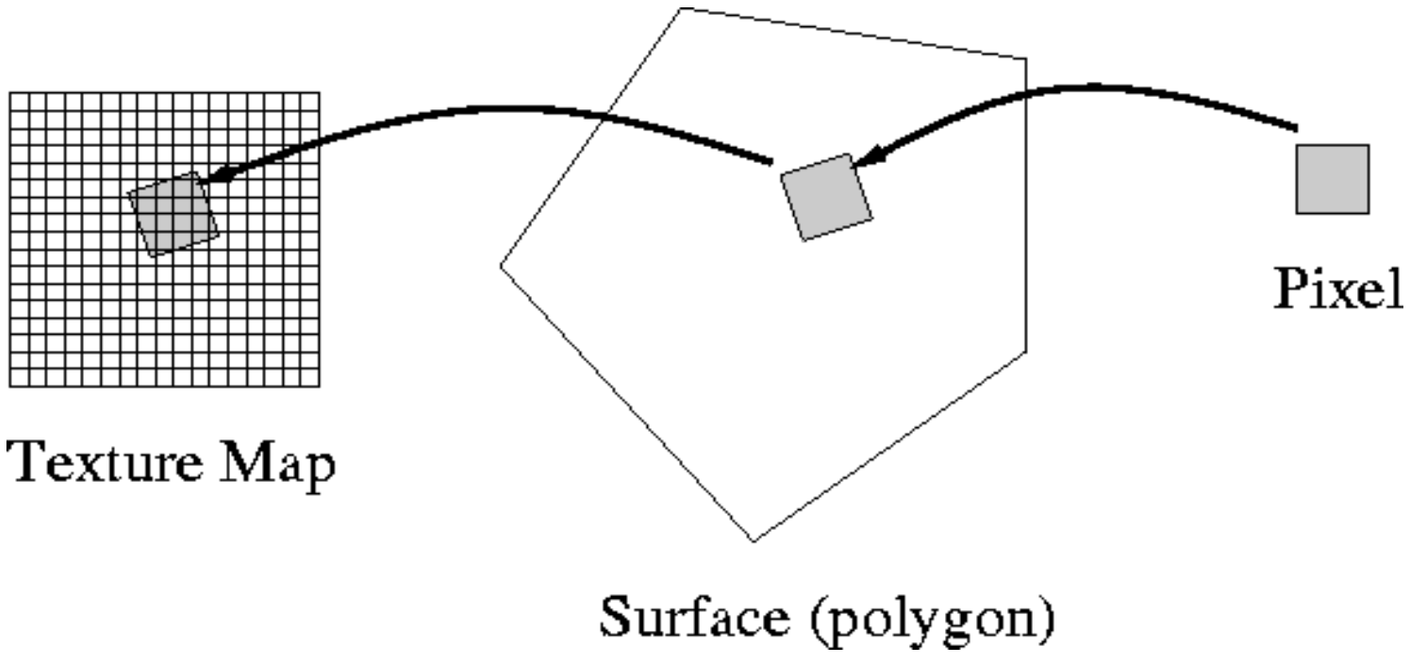
- *Specularity*
  - –used to generate a surface with variable shine.
- *Transparency*
  - –used to generate object with varying transparencies, like clouds. Used more to generate a complete object rather than just mapping the texture on the surface.
- Other than *environment mapping* the texture on the object is independent of the position of the object in the world. So *environment mapping* is distinct from *texture mapping*.

Problems

- Aliasing artifacts
  - Due to point sampling
- Perspective distortions
  - Since we did not take into account perspective transformation

Computing Color in Texture mapping

1. Associate texture with polygon
2. Map pixel onto polygon and then into texture map
3. Use weighted average of covered texture to compute color



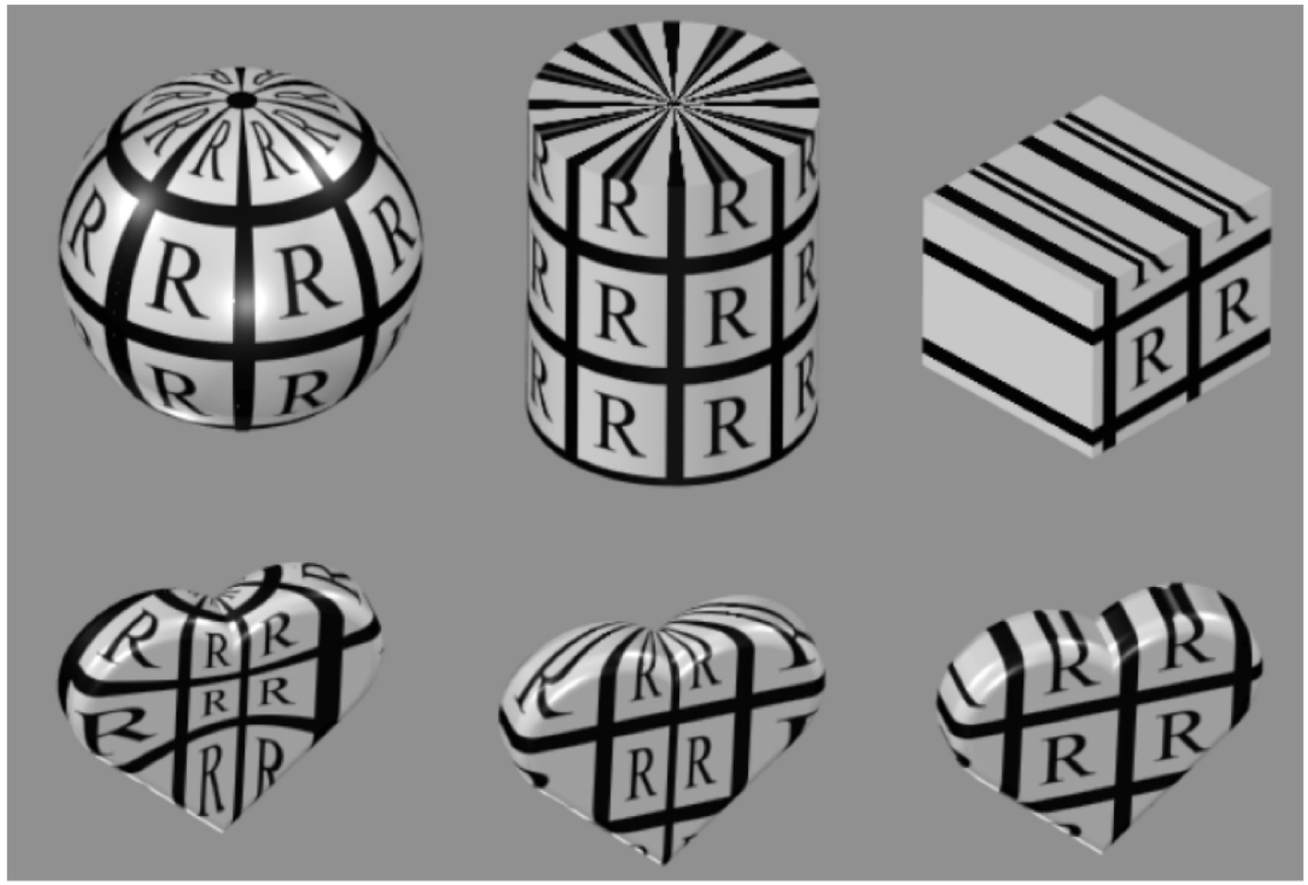
Projector Functions

- a method to generate texture coordinates,
  i. e. a method for surface parameterization
- planar projection

  $(x, y, z) \rightarrow (u, v)$
- cylindrical projection

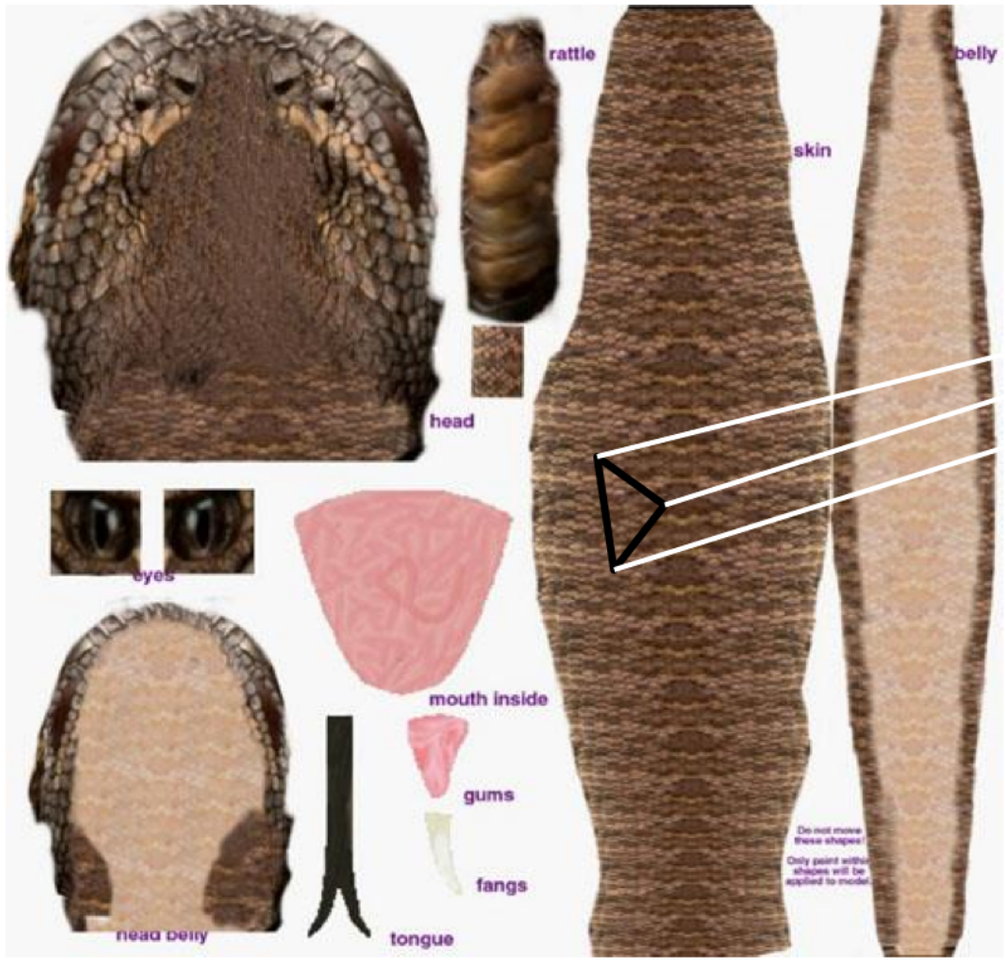
  $(x, y, z) \rightarrow (r, \theta, h) \rightarrow (u, v)$
- spherical projection

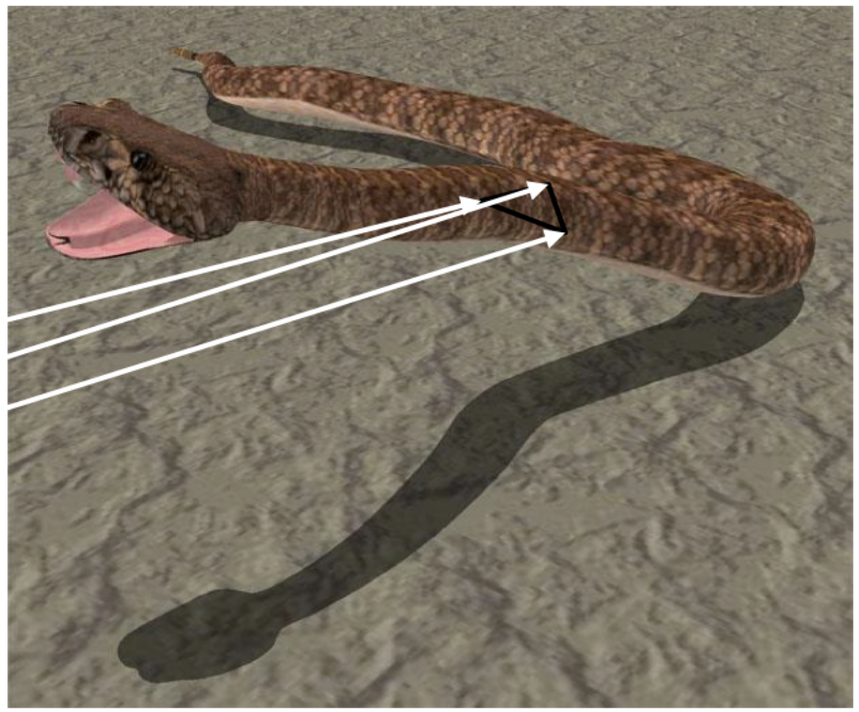
  $(x, y, z) \rightarrow (latitude, longitude) \rightarrow (u, v)$
- can be an initialization step for the surface parameterization



Projector Functions

- use of different projections for surface patches
  - minimize distortions, avoid artifacts at seams
  - several texture coordinates can be assigned to a single vertex





textured object

texture

Taking care of aliasing

- What causes aliasing artifacts
  - High frequency signals
- Typical solutions
  - Sample at higher rates
  - Pre-filter textures using low pass filters

Optimization

- Since most of the times, textures are known a priori, we can create various levels of these prefiltered textures in a preprocess.
- Then at run time, we fetch the required level of mipmap and apply that texture
- This is known as mipmapping

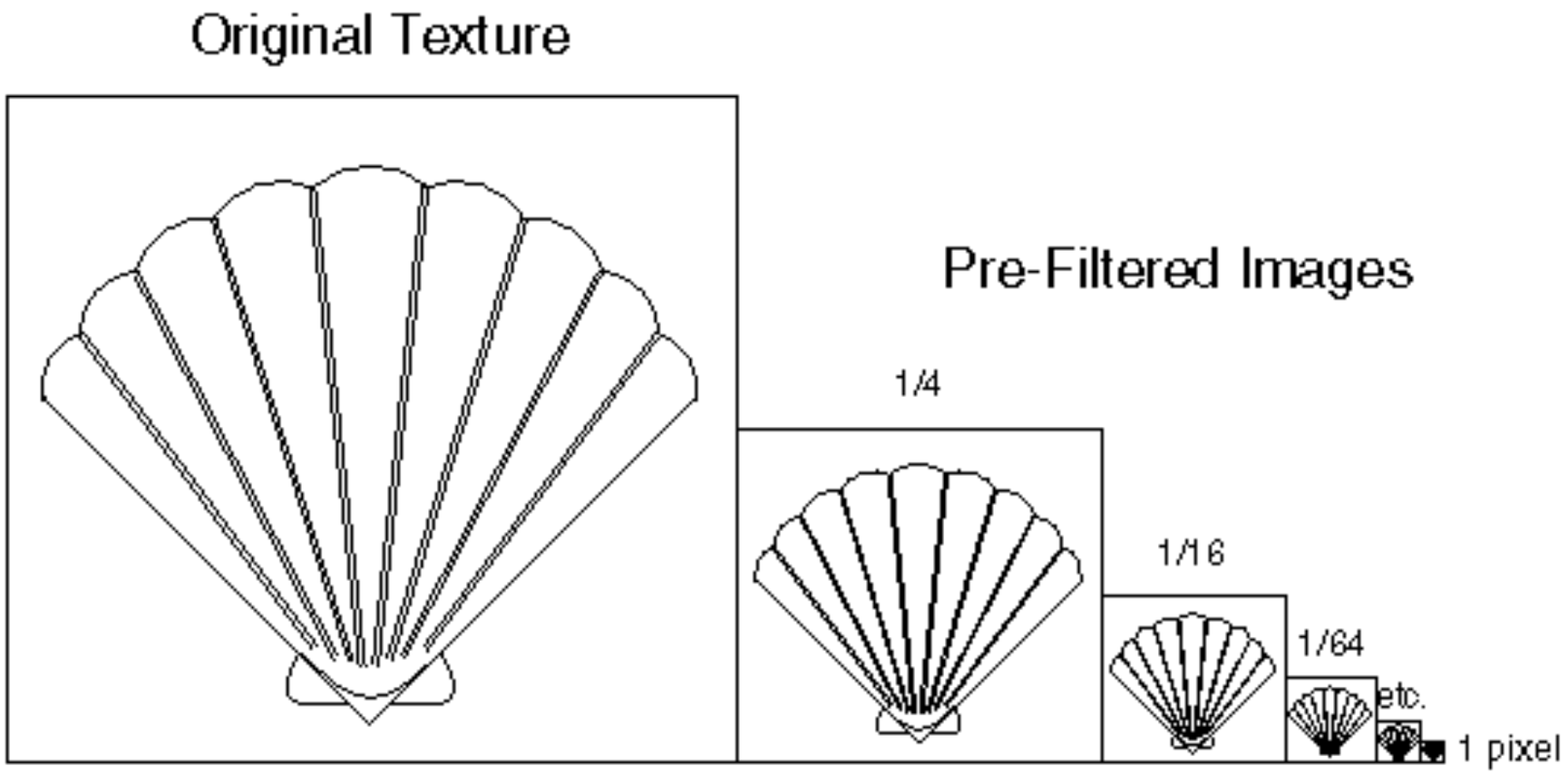
Mipmapping

- 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 (continued)

- One solution is to provide multiple levels of detail for the same texture and use the one that best matches the polygon's apparent size on screen.
- This technique is called Mipmapping, and each texture in the hierarchy is called a mipmap.
- OpenGL can compute mipmaps automatically, and it can also accept mipmaps provided from different files.

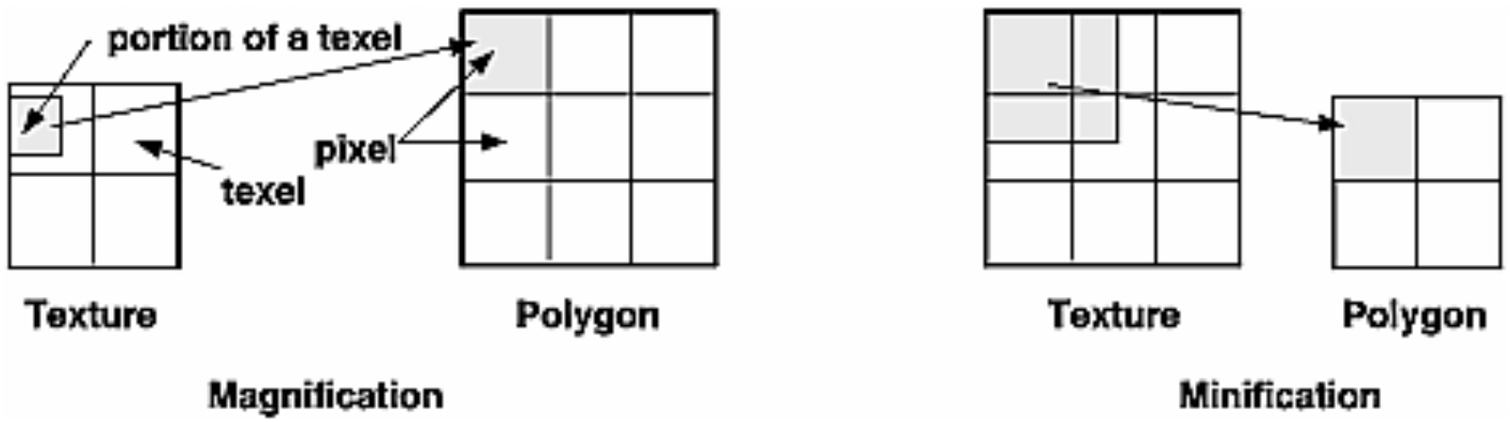
Mipmapping Example



Texture Filtering

- When the texture is mapped to a polygon, a single texel rarely matches a single pixel exactly.
- If a pixel matches only a portion of a texel, the texel must be magnified.
- If a pixel matches more than one texel, the texels must be minified.

Filtering Example

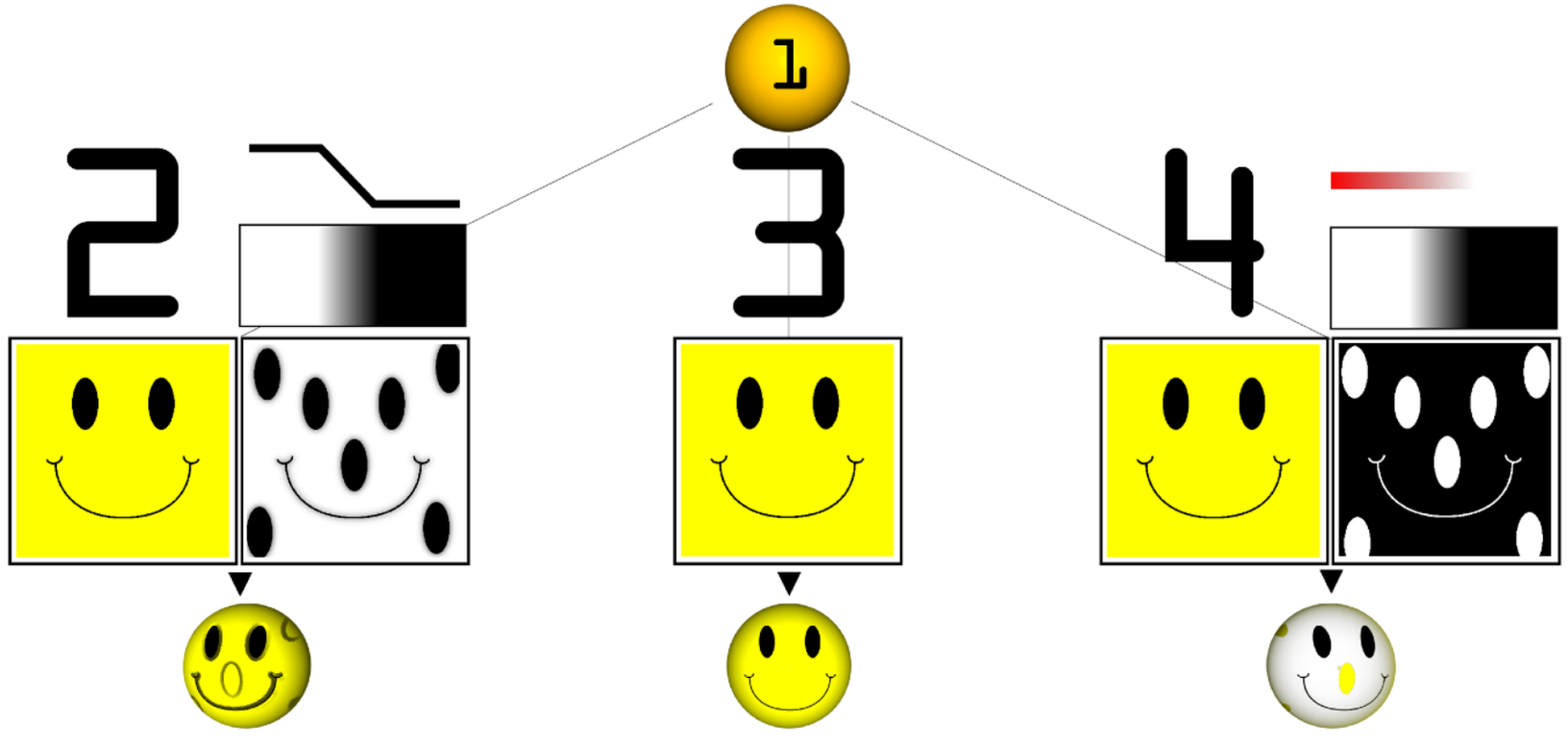


Texture Matrix

- Loading a texture onto the graphics card is very expensive
- But once there, a ***texture matrix*** can be used to "transform" the texture
  - For example, changing the translation can select different parts of the texture
- If the texture matrix is changed from frame to frame, the texture will appear to move on the object
- This is particularly useful for things like flame, or swirling vortices, …

Multi-texturing

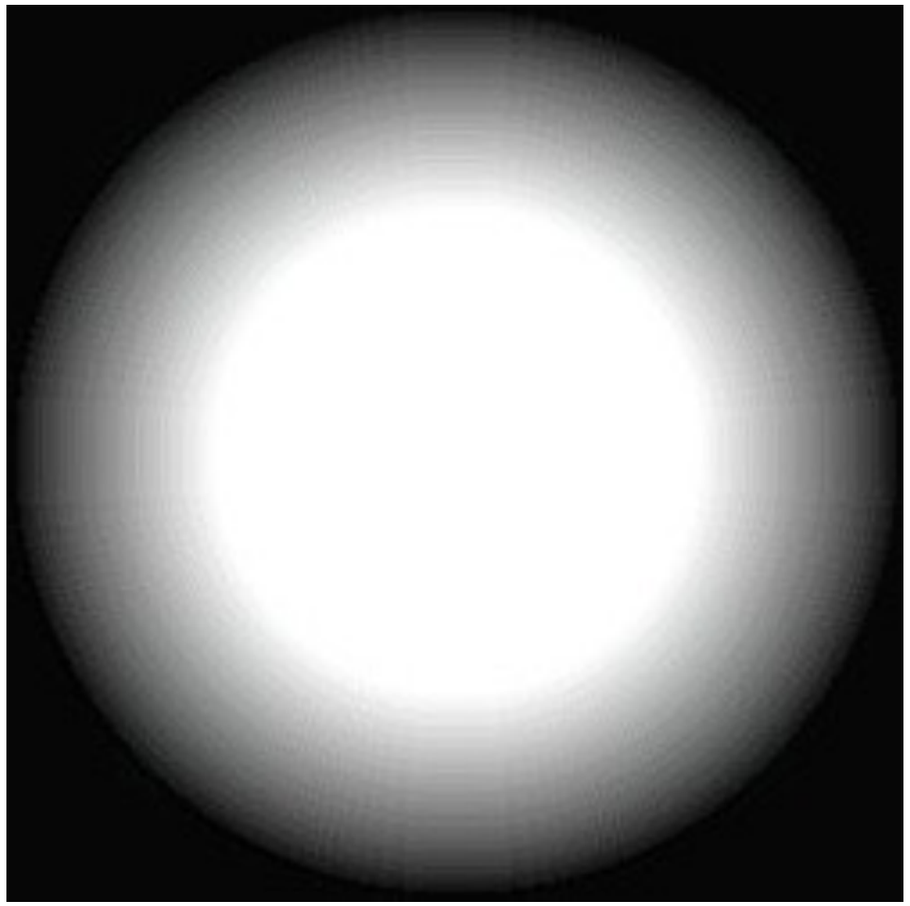
- It is sometimes possible to apply more than one texture to a polygon
- Examples: Light Maps, Texture Blending
- NOTE: implementations of OpenGL support multitexturing



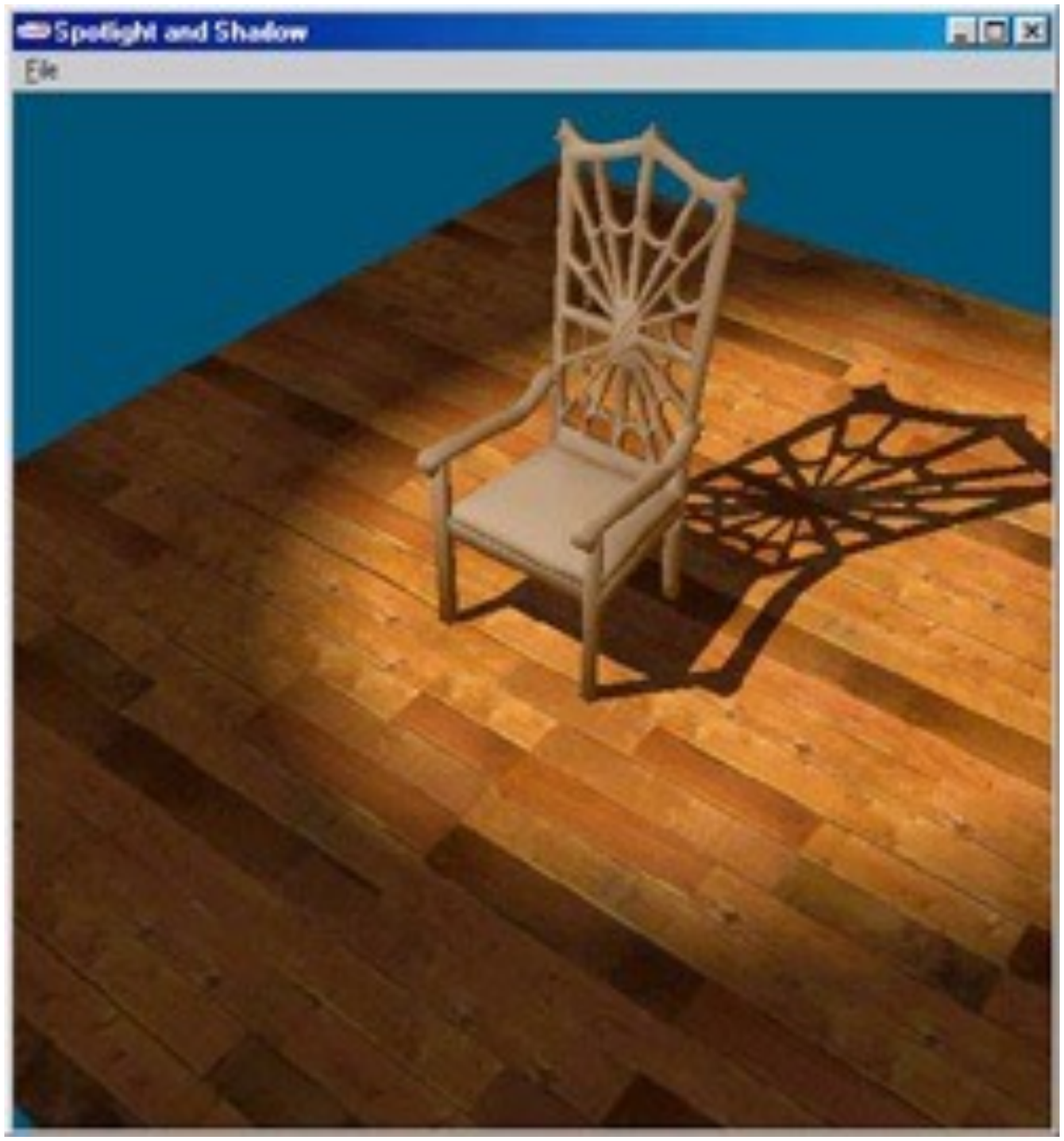
Light Maps

- 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.

Light Map Example



Light Map Example



Animated Textures

- Instead of a fixed image, it is sometimes possible to apply an animated clip (say, a Flash animation) as a texture.
- In this case, the actual frame of animation that is applied to the polygon as a texture changes with time.

OpenGL Texture mapping (Ref: Red book)

- Create texture objects
  - 1D, 2D, 3D
  - Specify what does texture contain: say, color, depth, etc.
- Specify how the texture is applied
  - Replace, modulate, or blend
- Enable texture mapping
- Specify scene with both geometric coordinates as well as texture coordinates

Specify Texture

glTexImage2D (target, level, internalformat, width, height, border, format, type, *texels);

- E.g.:
  - target: GL_TEXTURE_2D
  - Level : 0 (for no mip maps)
  - Internalformat: GL_RGB
  - Width,height: 2^m+2*b, 256 by 256 with b=0
  - Border b: 0
  - Format: GL_RGB
  - Type: GL_INT
  - texels: actual texture image

Specifying mipmaps

- You need to specify the levels for which mipmapping is being used.
- You can also specify scale factor (between the texture image and size of polygon).
- You also need to supply images for all levels.
- You use glTexParameteri() and glTexImage2D()
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_BASE_LEVEL, 2);
  - glTexImage2D(..,2,…)

Automatic mipmap generation

- If you have highest resolution image, OpenGL can generate low level prefiltered images automatically.
- gluBuild2DMipmaps()
- gluBuild2DMipmapsLevels()
  - To build only a subset
- See book for detailed description of parameters

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);

Other calls

- glGenTextures();
  - To generate unique names

- glBindTexture();
  - Creates if texture object not already created
  - Activates the texture object if already created
  - Pass 0 to stop using texture objects

- glEnable(GL_TEXTURE_2D);
  - Enables texture mapping

- glTexCoord2i(texcoordinate);

Modulating surface properties

- 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);

Tricks with Textures

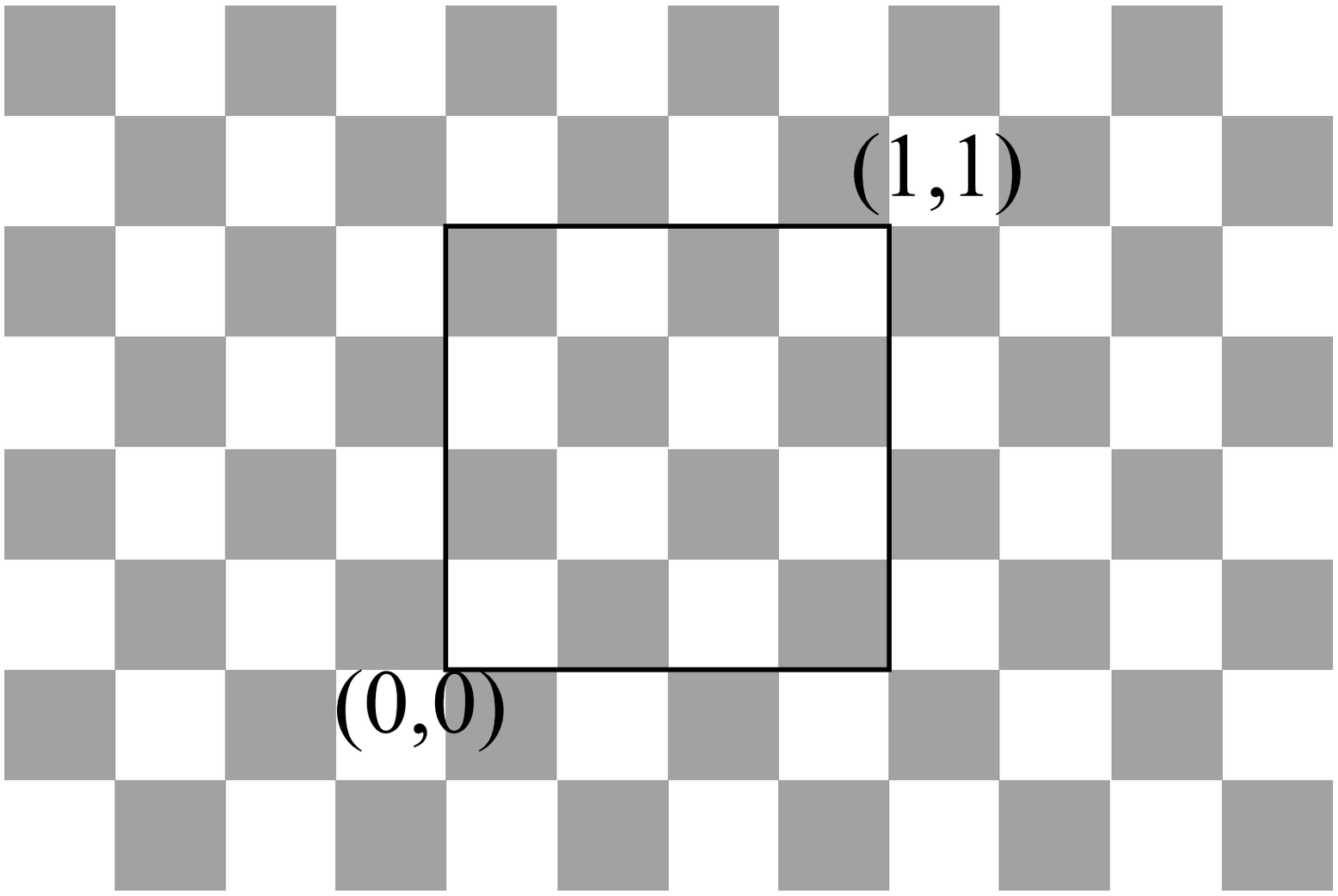
- If the texture is not large enough to cover a polygon, you can tile it using the GL_REPEAT parameter
  - glTexParameteri (GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);

- You can also clamp the texture (i.e., stretch the last pixel to cover everything)

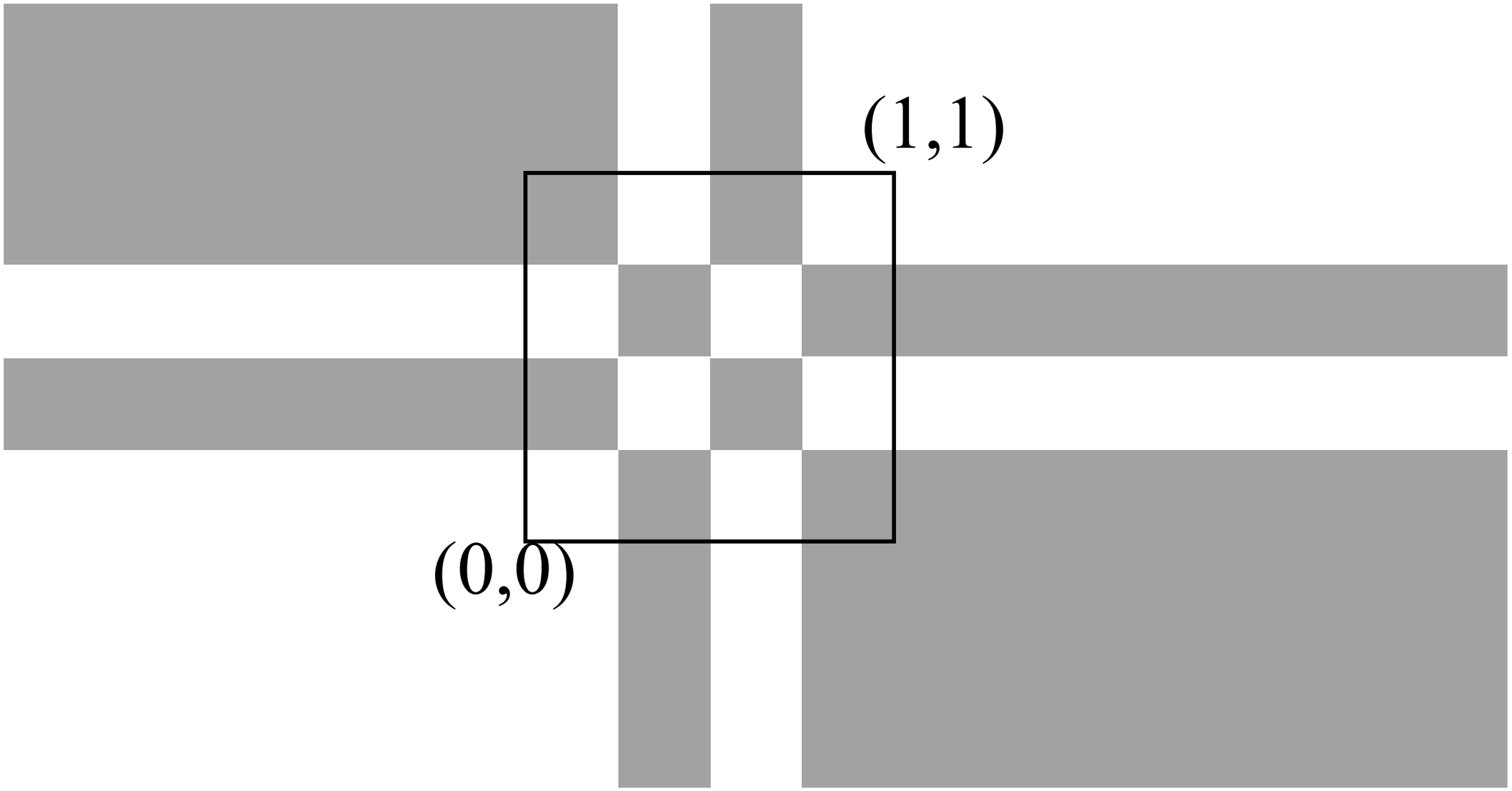
Boundaries

- 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)

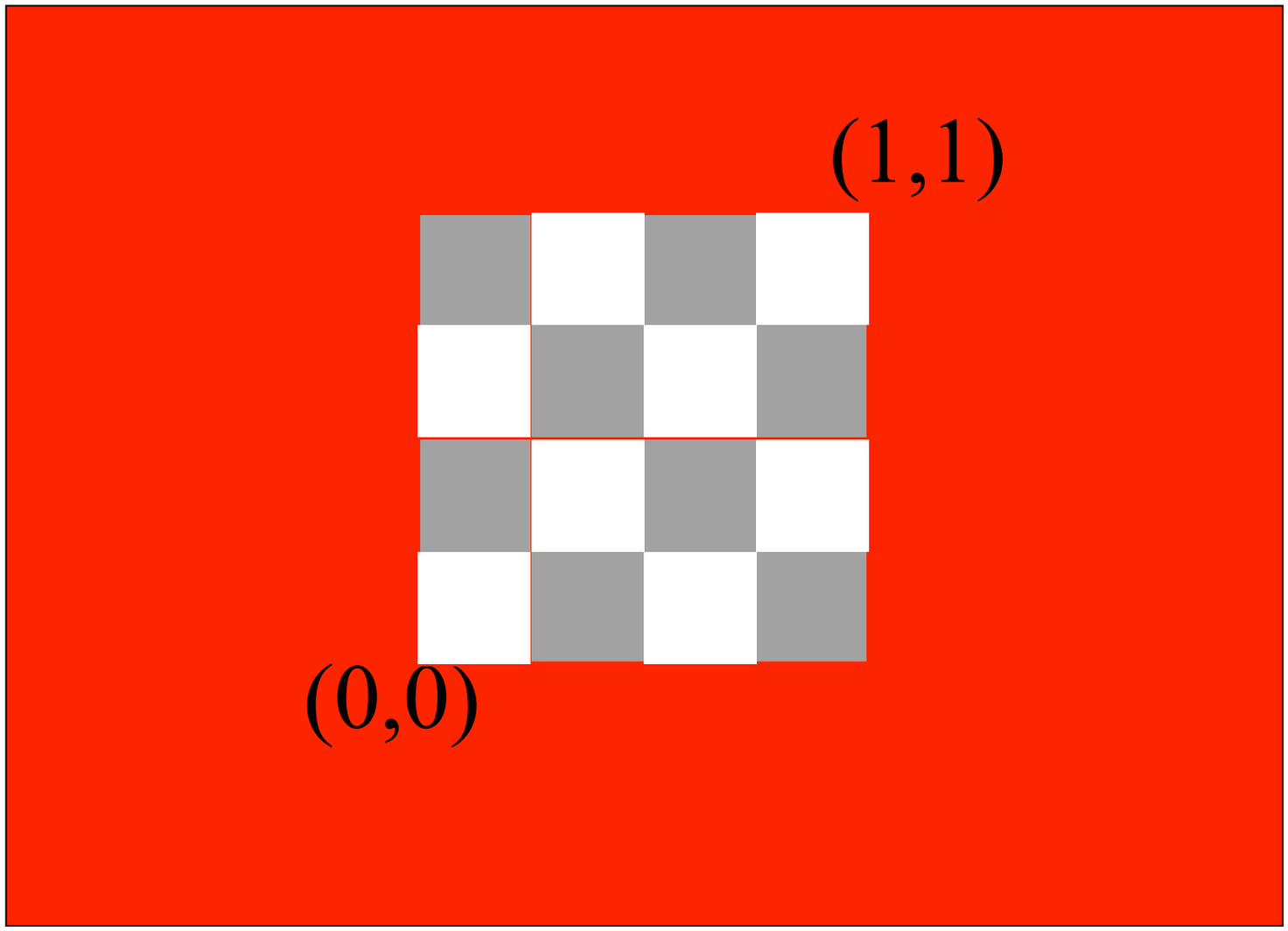
Repeat Border



Clamp Border



Border Color



Other Texture Stuff

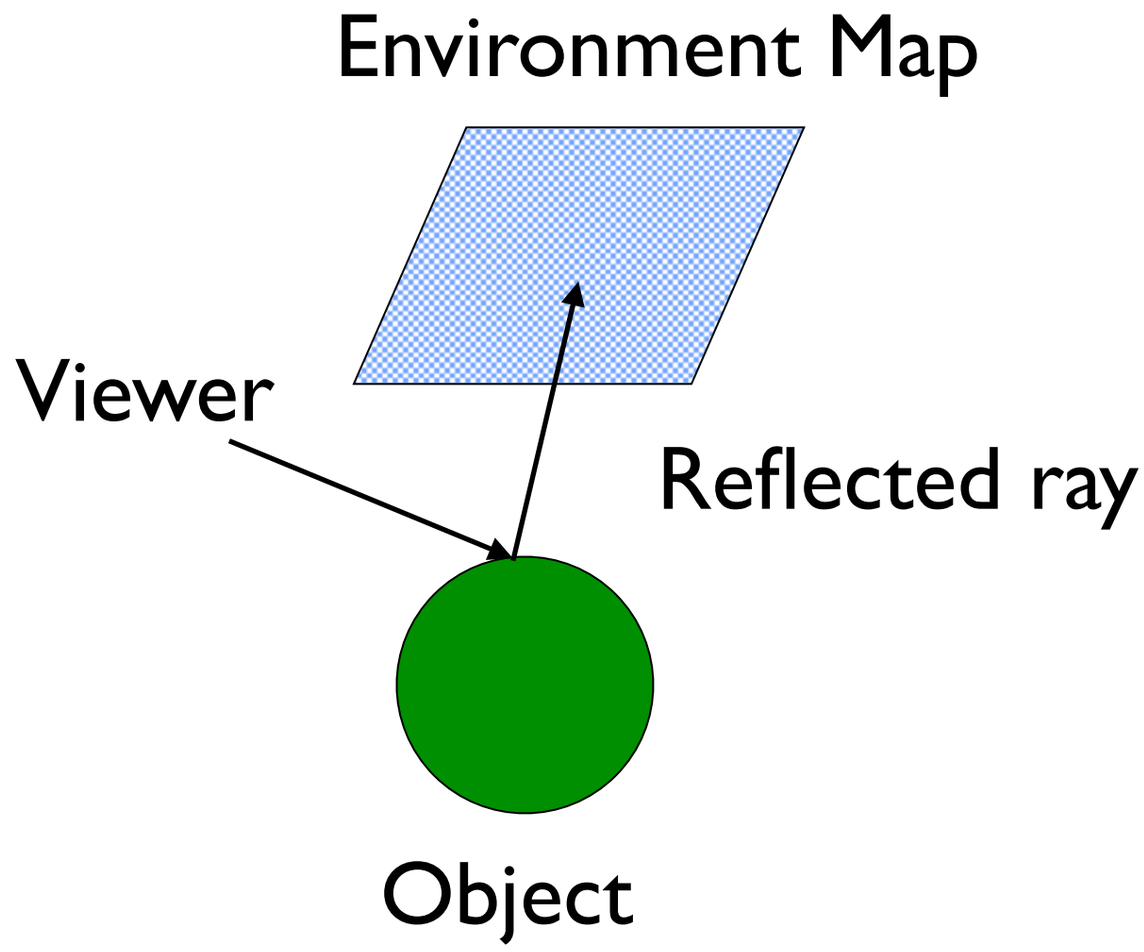
- Texture must be in fast memory - it is accessed for every pixel drawn
  - If you exceed it, performance will degrade horribly
  - There are functions for managing texture memory
    - Skilled artists can pack textures for different objects into one map
- Texture memory is typically limited, so a range of functions are available to manage it
- Specifying texture coordinates can be annoying, so there are functions to automate it for specific shapes, say quadrics, NURBS

Yet More Texture Stuff

- There is a 4x4 texture matrix: apply a matrix transformation to texture coordinates before indexing texture
- There are "image processing" operations that can be applied to the pixels coming out of the texture

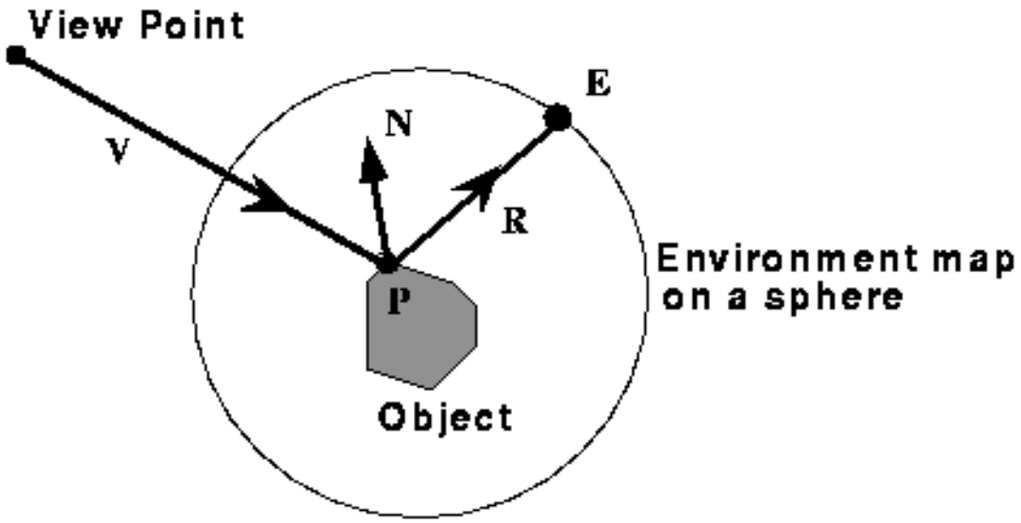
Environment Mapping

- 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:
  $\boldsymbol{R}=2(\boldsymbol{N}\cdot\boldsymbol{V})\boldsymbol{N}-\boldsymbol{V}$



Environment Maps

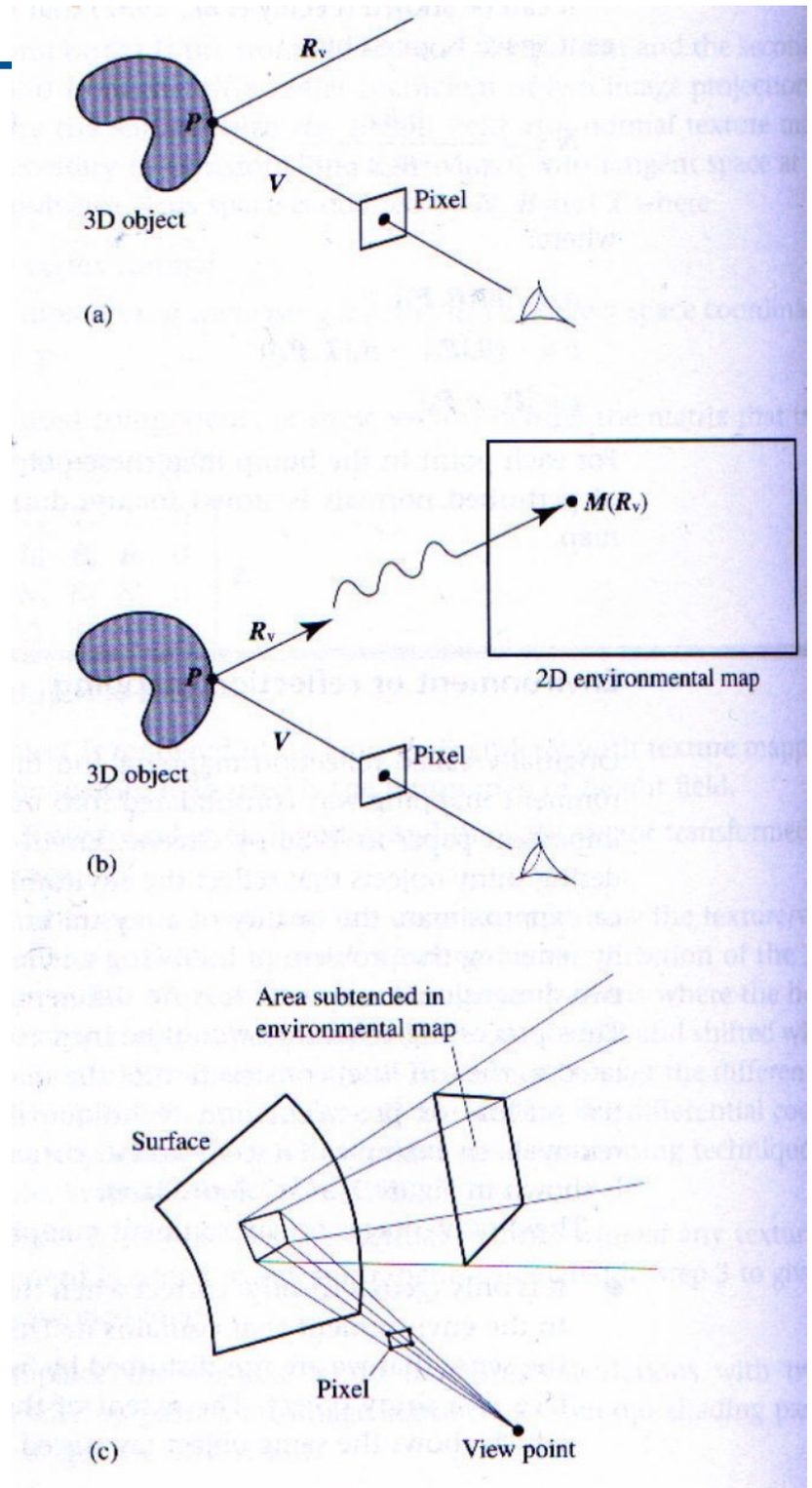
Environment Maps



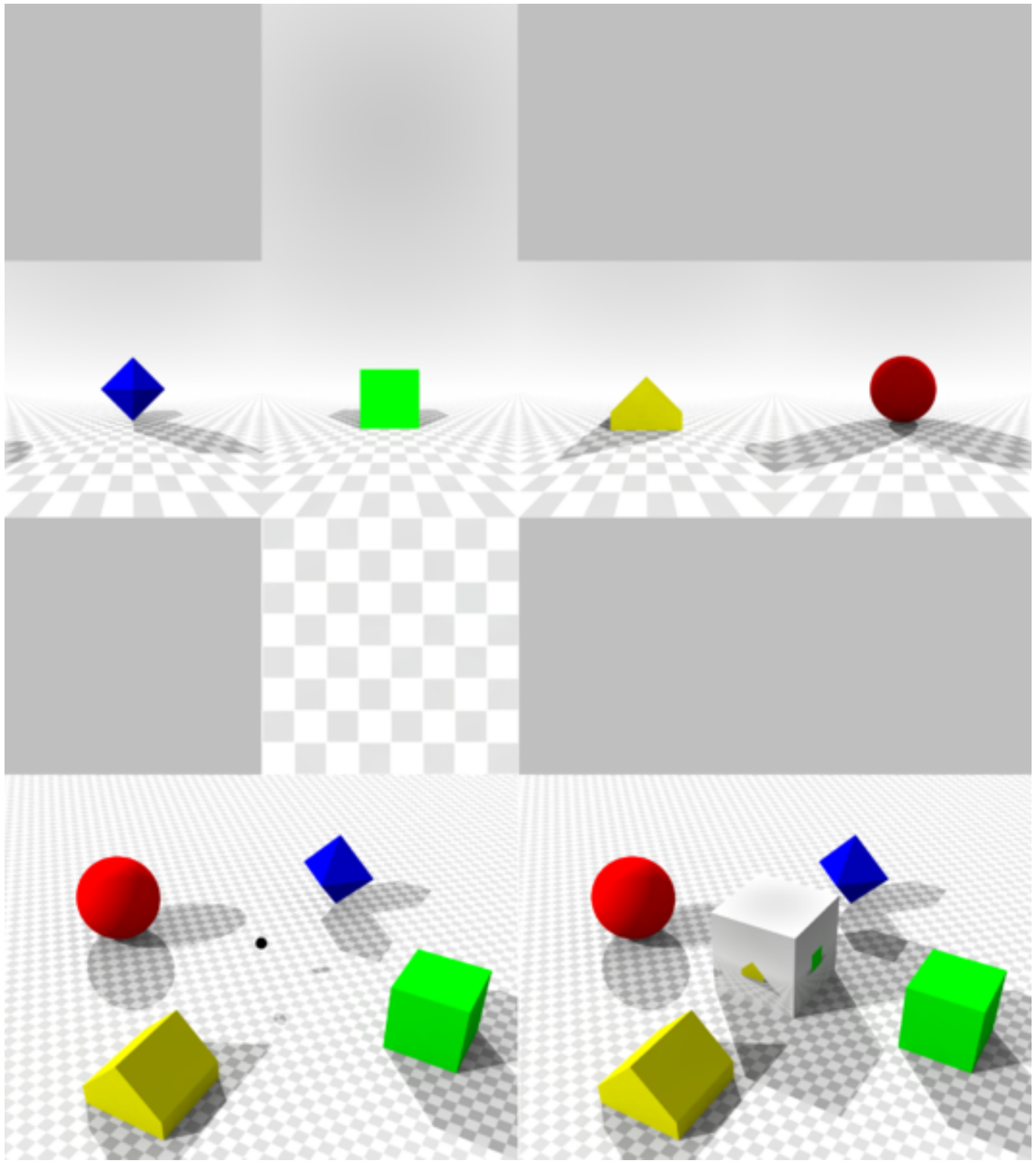
- We use the *direction* of the reflected ray to index a texture map.
- We can simulate reflections. This approach is not completely accurate. It assumes that all reflected rays begin from the same point, and that all objects in the scene are the same distance from that point.

Sphere Mapping

- The map lives on a sphere, but now the coordinate mapping is simplified
- We use latitude and longitude as (u,v)



Cube Mapping



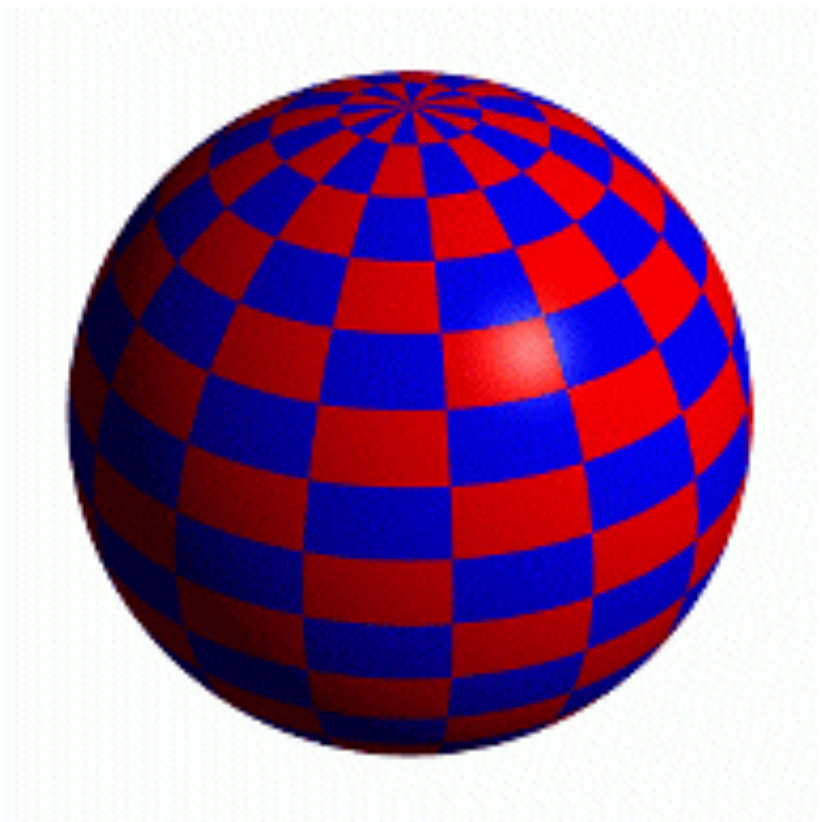
Cube Mapping

- The map resides on the surfaces of a cube around the object
  - Typically, align the faces of the cube with the coordinate axes

- To generate the map:
  - For each face of the cube, render the world from the center of the object with the cube face as the image plane
    - Rendering can be arbitrarily complex (it's off-line)
  - Or, take 6 photos of a real environment with a camera in the object's position
- Assume the cube's faces are aligned with the coordinate axes, and have texture coordinates in [0,1]x[0,1]. Main issue is to decide
  - which face to use, and
  - decide which texture coordinates to use.
  - Use a ray from the centre; only cube corners and edges will need special attention

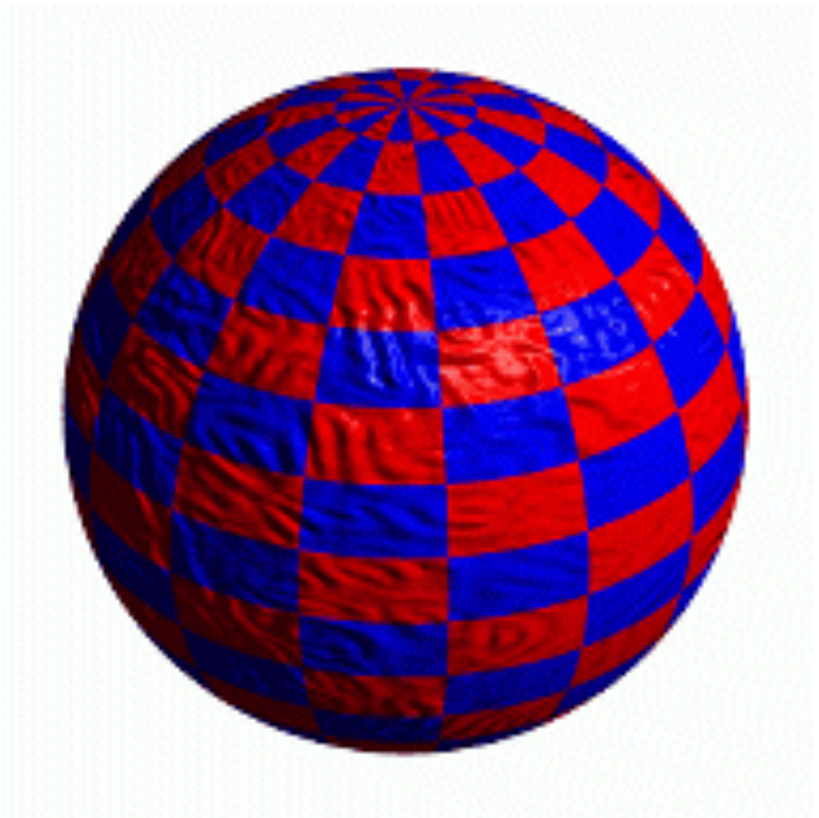
Bump Mapping

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



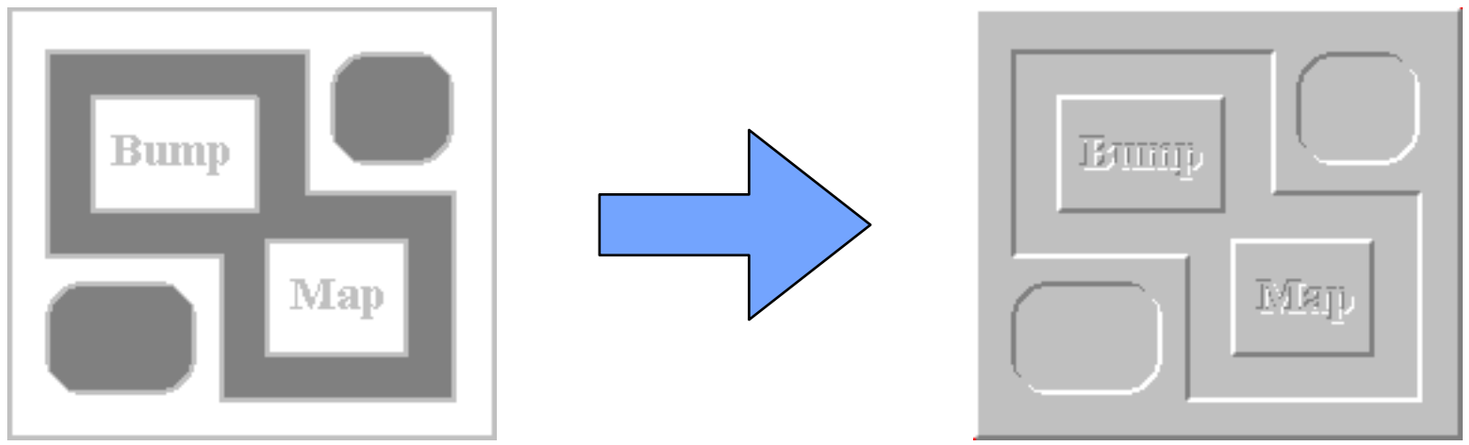
Sphere w/Diffuse Texture & Bump Map

Bump mapping

REF： http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm

Bump mapping

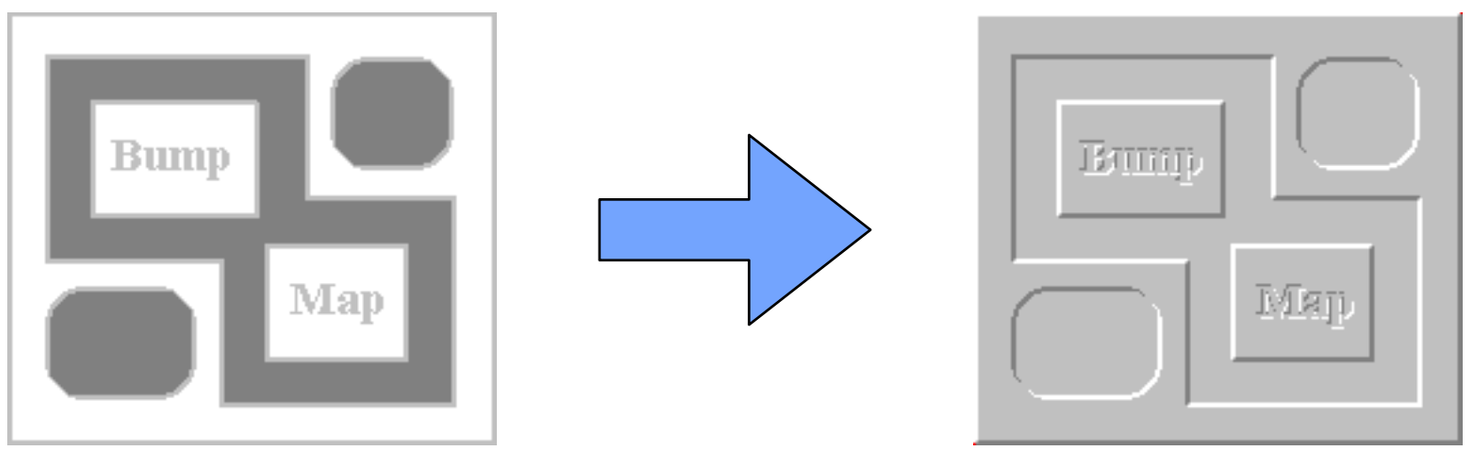
REF： http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm



Bump mapping

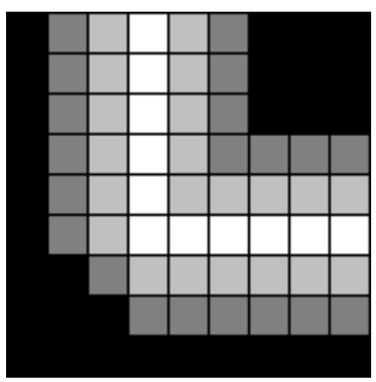
- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.

REF： http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm



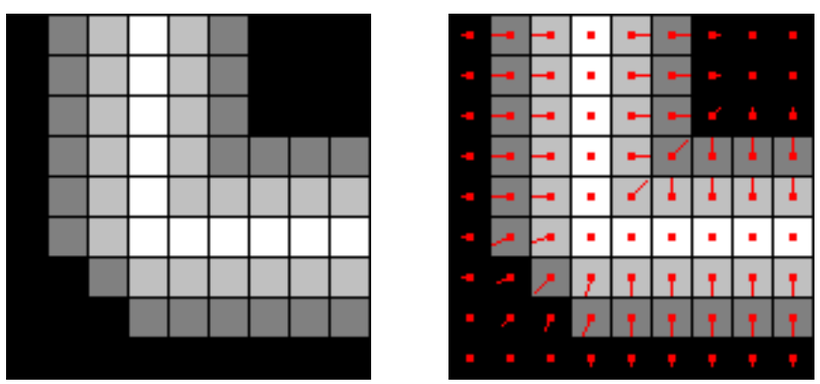
Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.



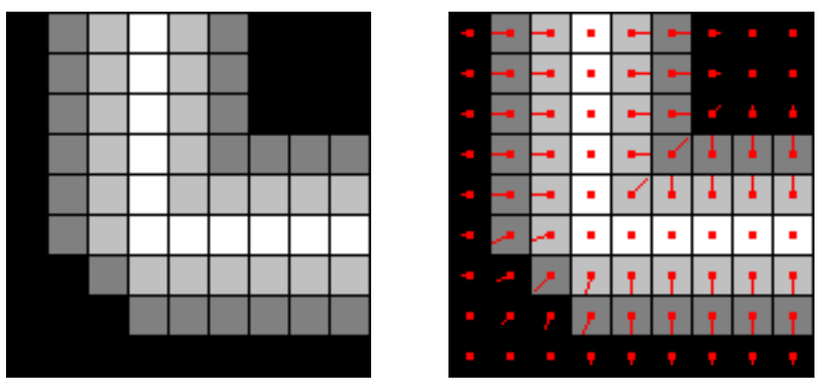
Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.



Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.

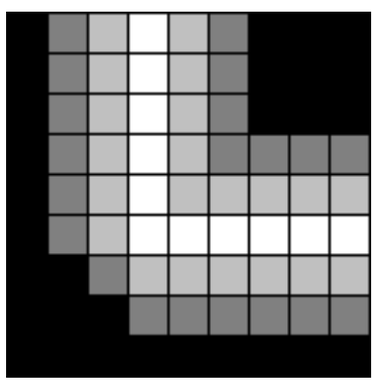


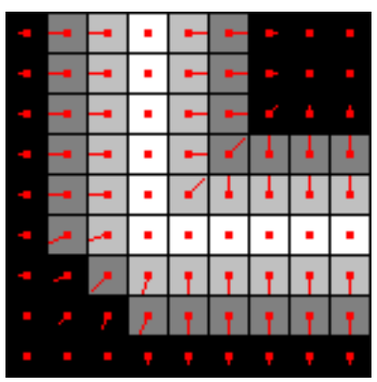
x_gradient = pixel(x-1, y) - pixel(x+1, y)

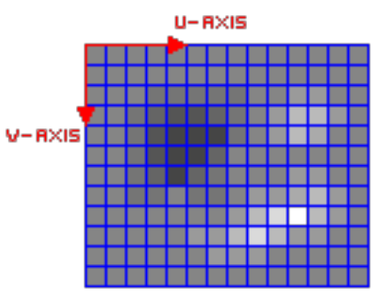
y_gradient = pixel(x, y-1) - pixel(x, y+1)

Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.





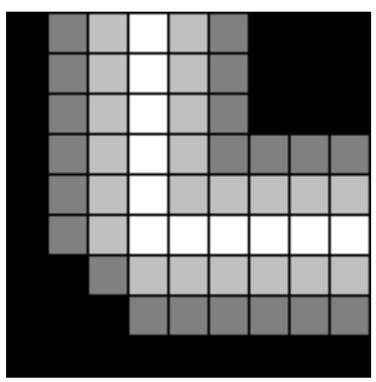


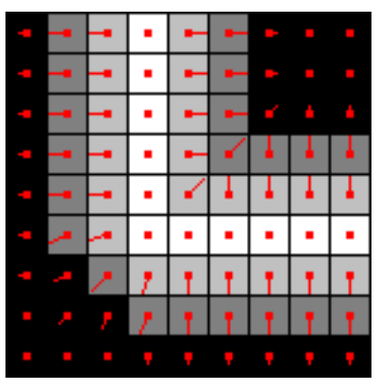
x_gradient = pixel(x-1, y) - pixel(x+1, y)

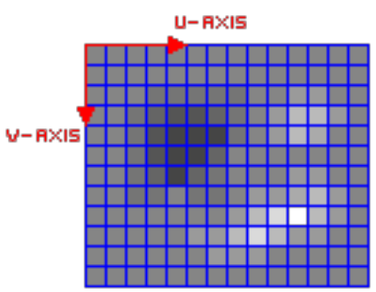
y_gradient = pixel(x, y-1) - pixel(x, y+1)

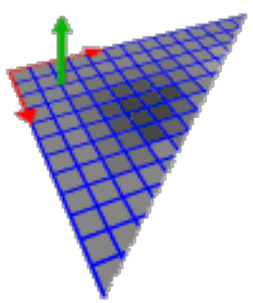
Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.







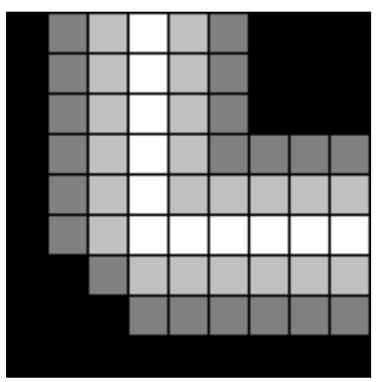
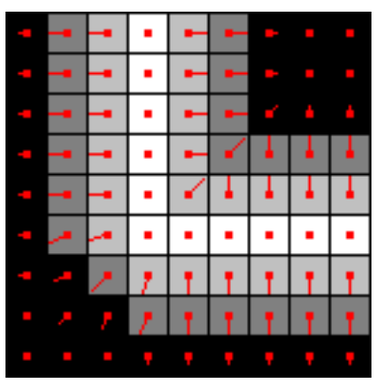
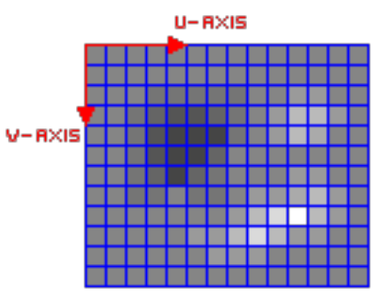


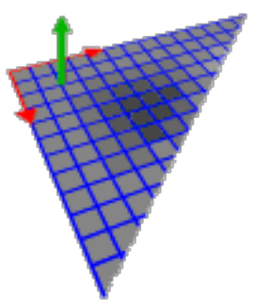
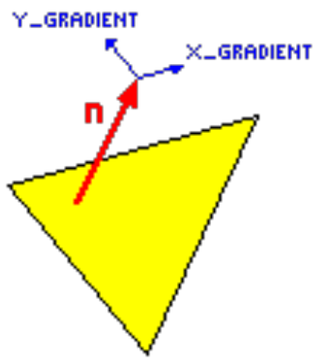
x_gradient = pixel(x-1, y) - pixel(x+1, y)

y_gradient = pixel(x, y-1) - pixel(x, y+1)

Bump mapping

- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.

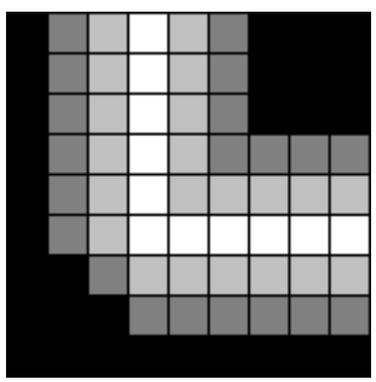



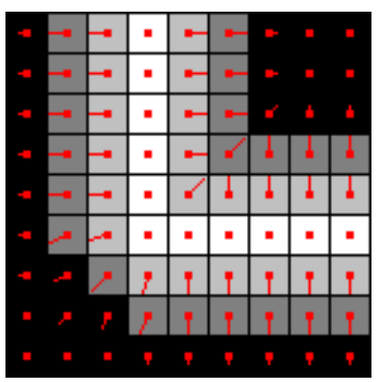
x_gradient = pixel(x-1, y) - pixel(x+1, y)

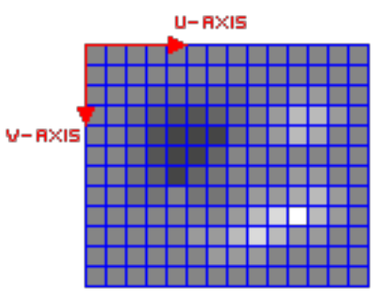
y_gradient = pixel(x, y-1) - pixel(x, y+1)

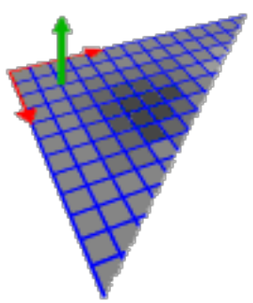
Bump mapping

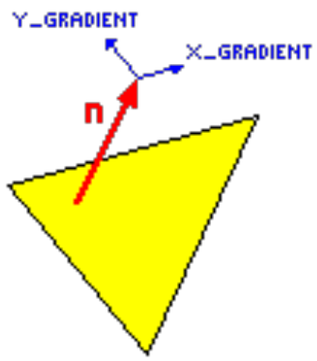
- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.









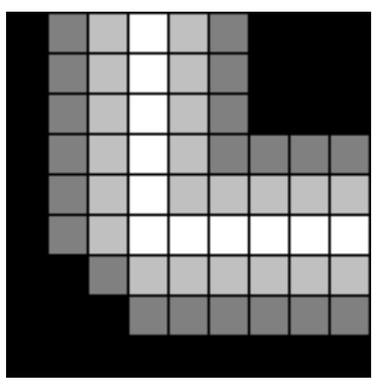


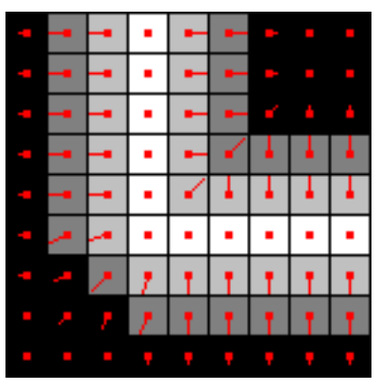
x_gradient = pixel(x-1, y) - pixel(x+1, y)
y_gradient = pixel(x, y-1) - pixel(x, y+1)

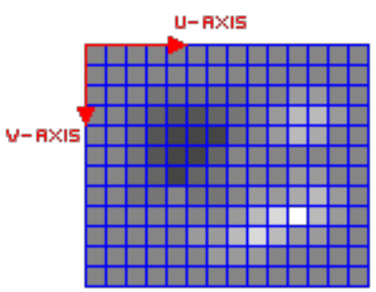
New_Normal = Normal + (U * x_gradient) + (V * y_gradient)

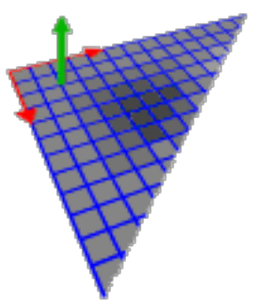
Bump mapping

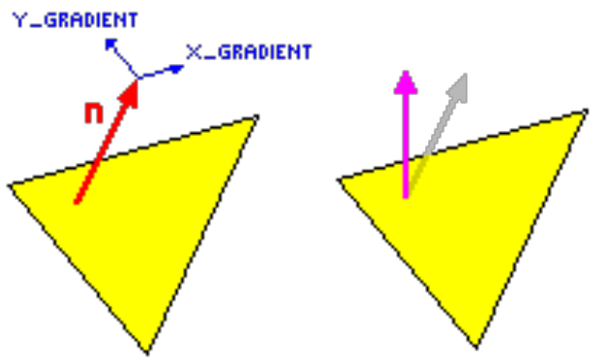
- Before lighting a calculation is performed for each visible point (or pixel) on the object's surface:
  1. Look up the height in the heightmap that corresponds to the position on the surface.
  2. Calculate the surface normal of the heightmap, typically using the finite difference method.
  3. Combine the surface normal from step two with the true ("geometric") surface normal so that the combined normal points in a new direction.
  4. Calculate the interaction of the new "bumpy" surface with lights in the scene using, for example, the Phong reflection model.









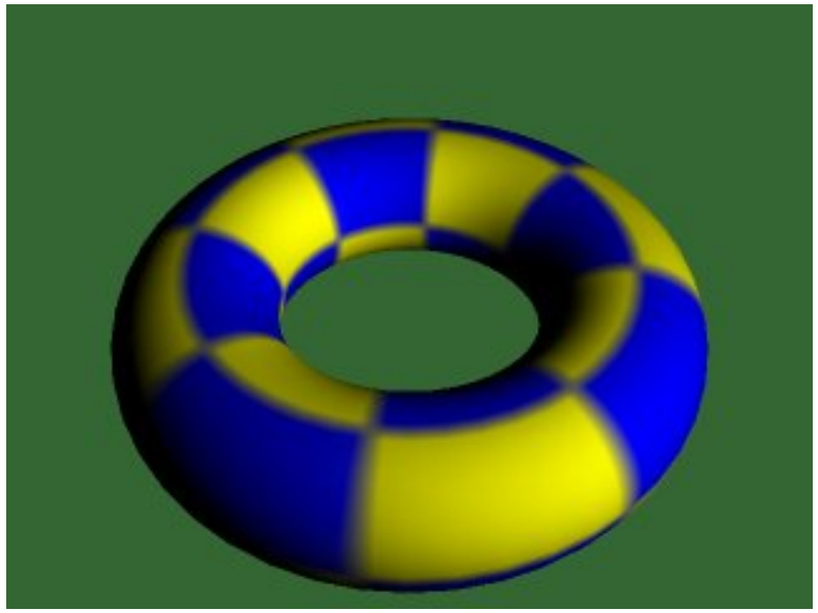
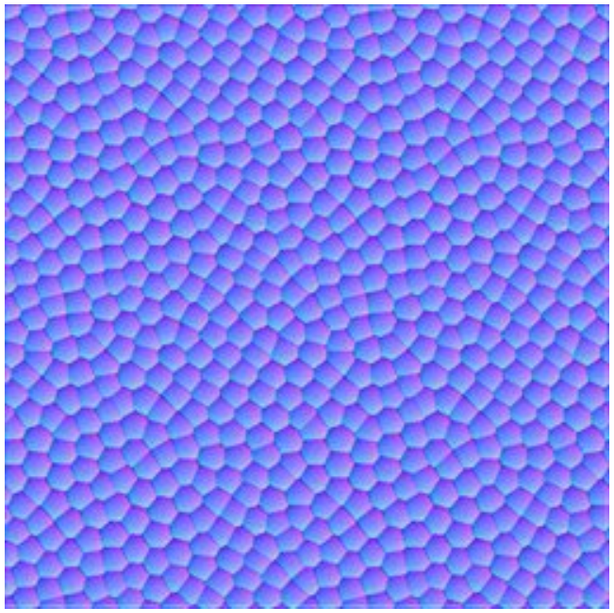
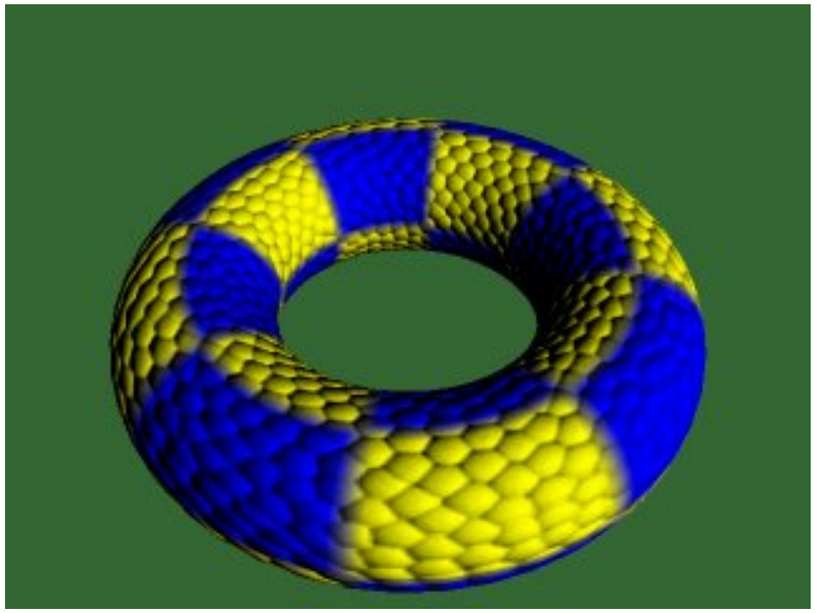


x_gradient = pixel(x-1, y) - pixel(x+1, y)
y_gradient = pixel(x, y-1) - pixel(x, y+1)

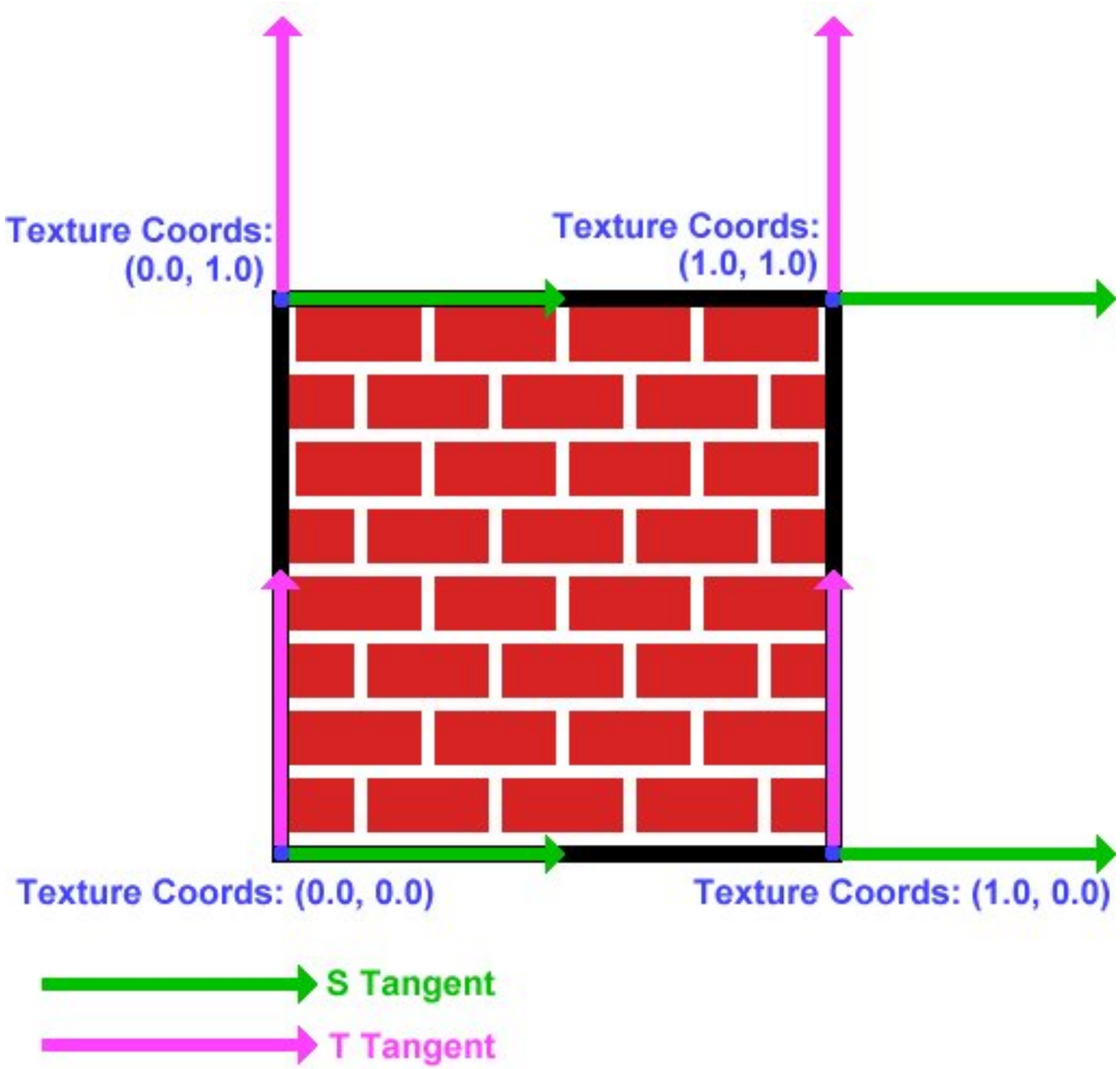
New_Normal = Normal + (U * x_gradient) + (V * y_gradient)

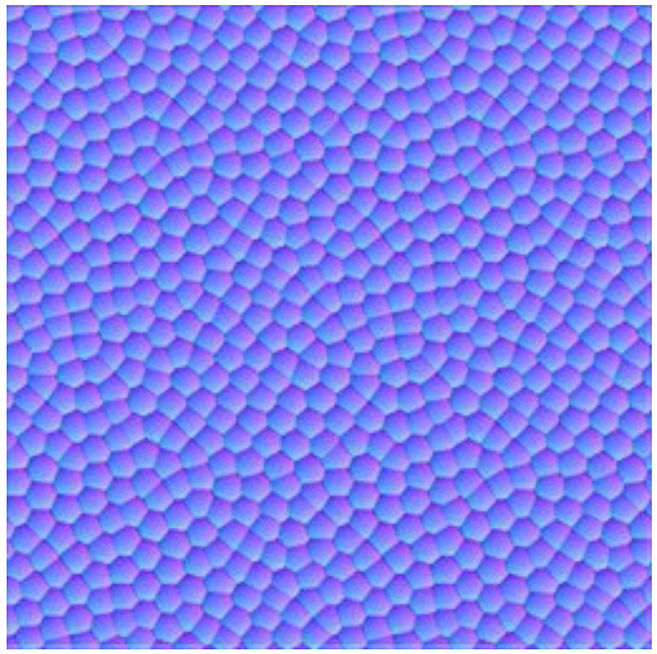
Normal map

- Simple Bump mapping
  - http://www.paulsprojects.net/tutorials/simplebump/simplebump.html

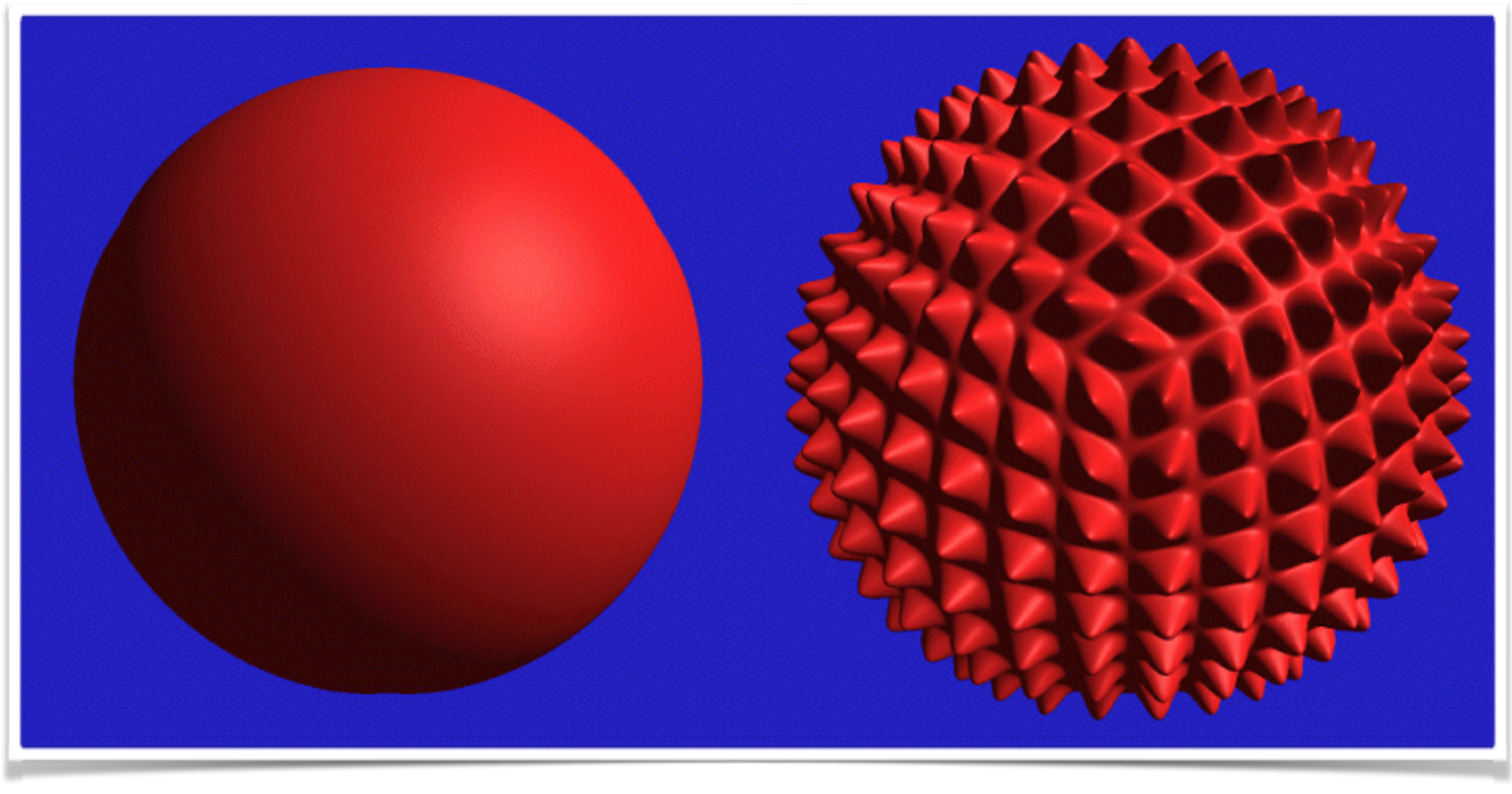
Normal map


Texture Coords:
(0.0, 1.0)
Texture Coords:
(1.0, 1.0)
Texture Coords: (0.0, 0.0)
Texture Coords: (1.0, 0.0)
S Tangent
T Tangent



Displacement Mapping

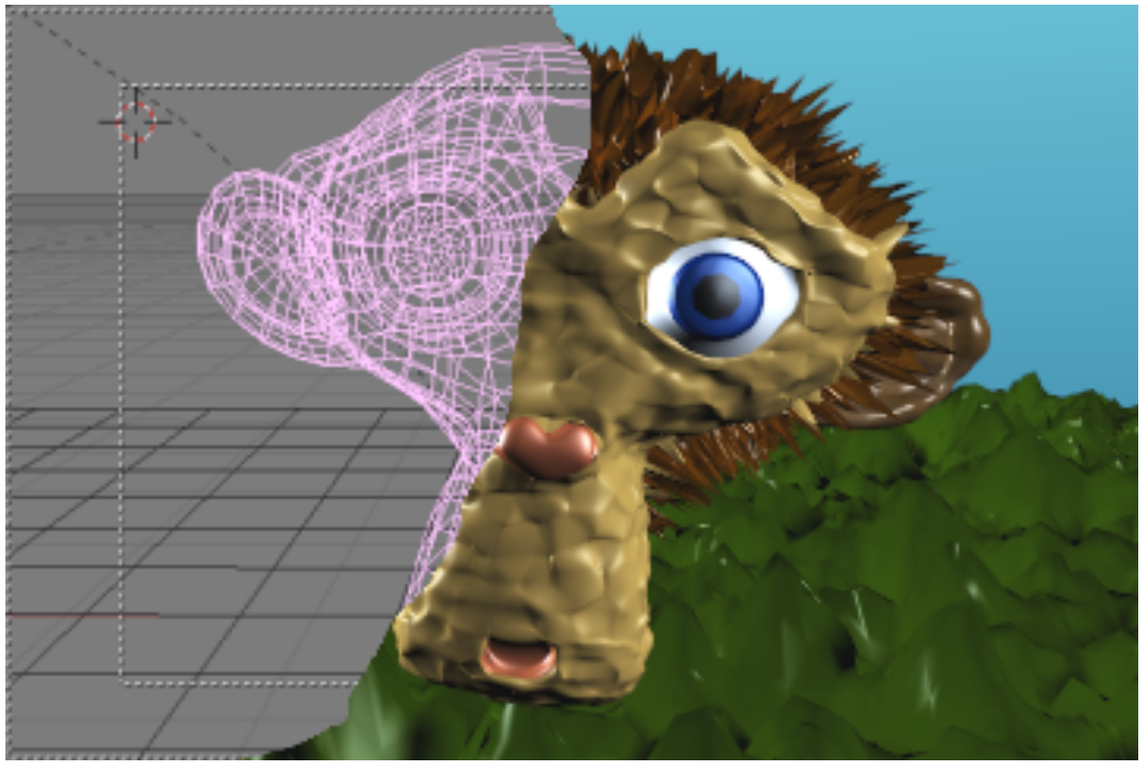
- use the texture map to actually move the surface point
  - [Cook 1984]
  - How is this fundamentally different than bump mapping?

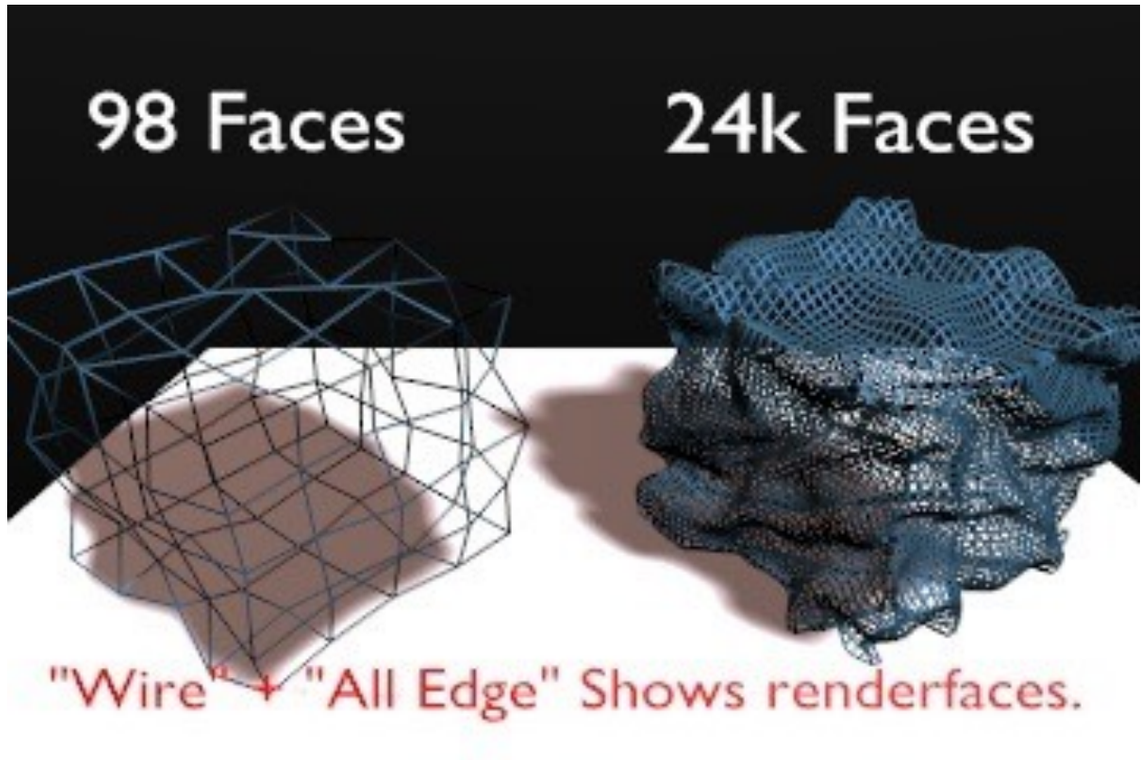


The geometry must be displaced before visibility is determined.

Displacement Mapping

- use the texture map to actually move the surface point
  - [Cook 1984]

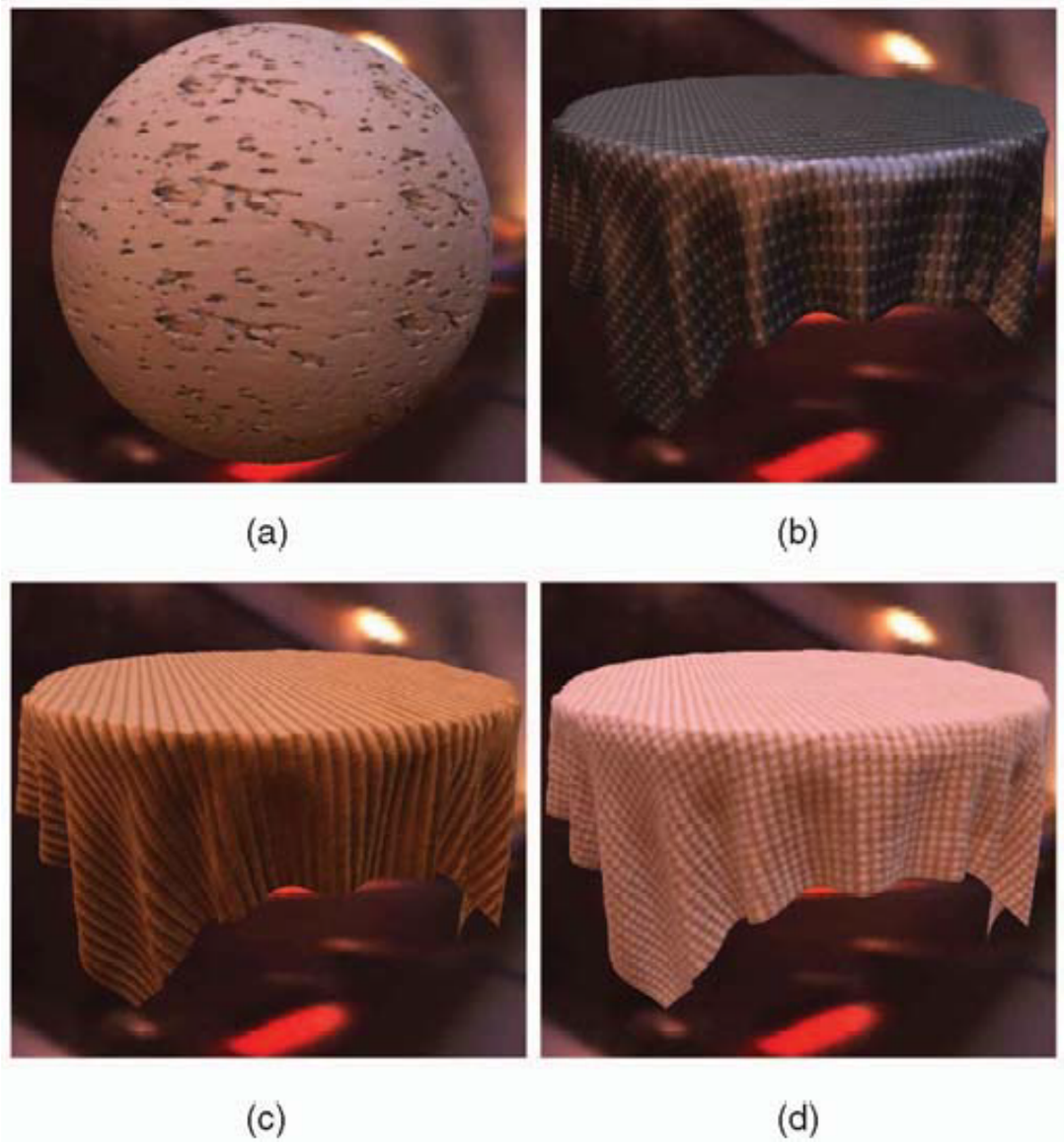


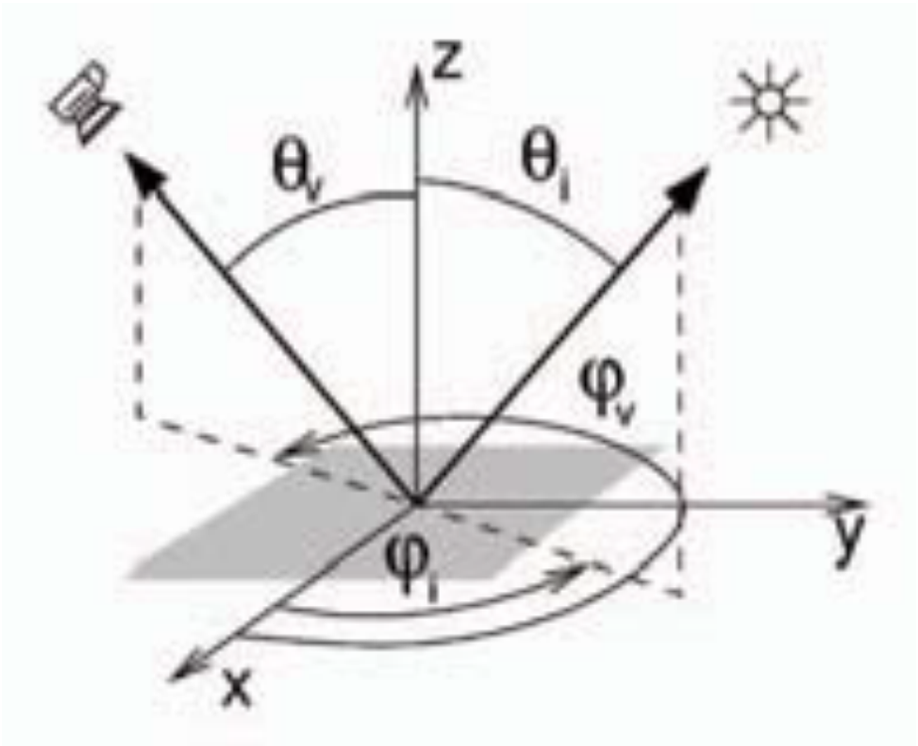


Procedural Texture Mapping

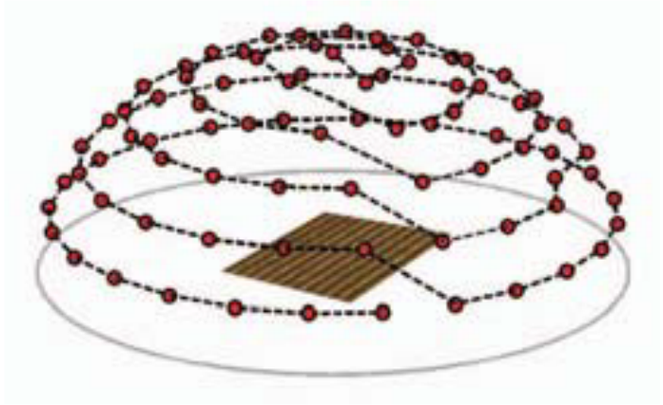
- Instead of looking up an image, pass the texture coordinates to a function that computes the texture value on the fly
  - Renderman, the Pixar rendering language, does this
  - Available in a limited form with vertex shaders on current generation hardware
- Advantages:
  - Near-infinite resolution with small storage cost
  - Idea works for many other things
- Has the disadvantage of being slow in many cases

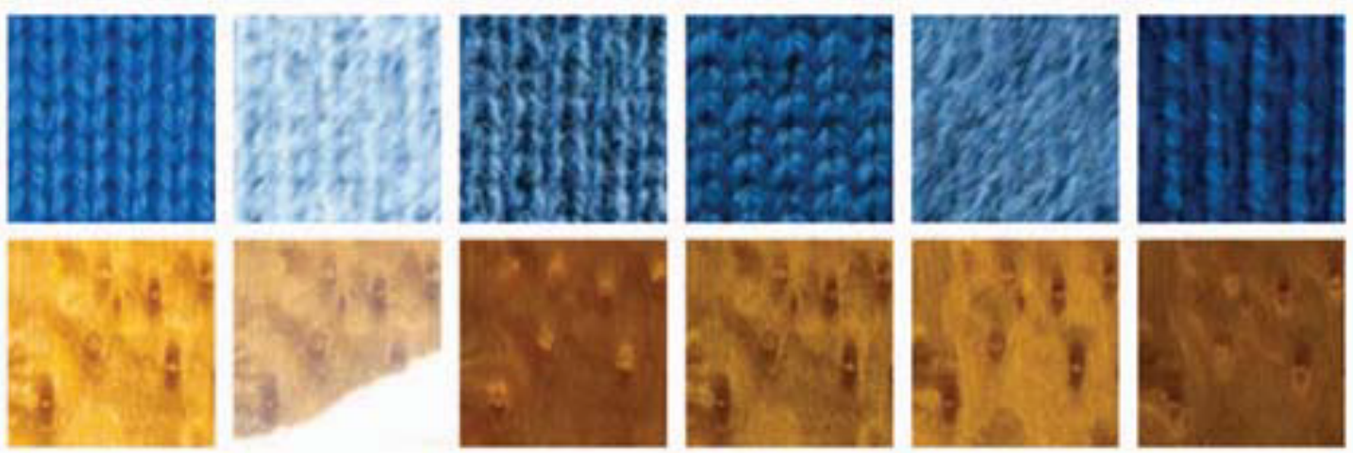
Bidirectional texture function





Bidirectional texture function





THANK YOU

- REF:
  - CMU: SURVEY OF TEXTURE MAPPING
  - http://cg.informatik.uni-freiburg.de/course_notes/graphics_06_texturing.pdf