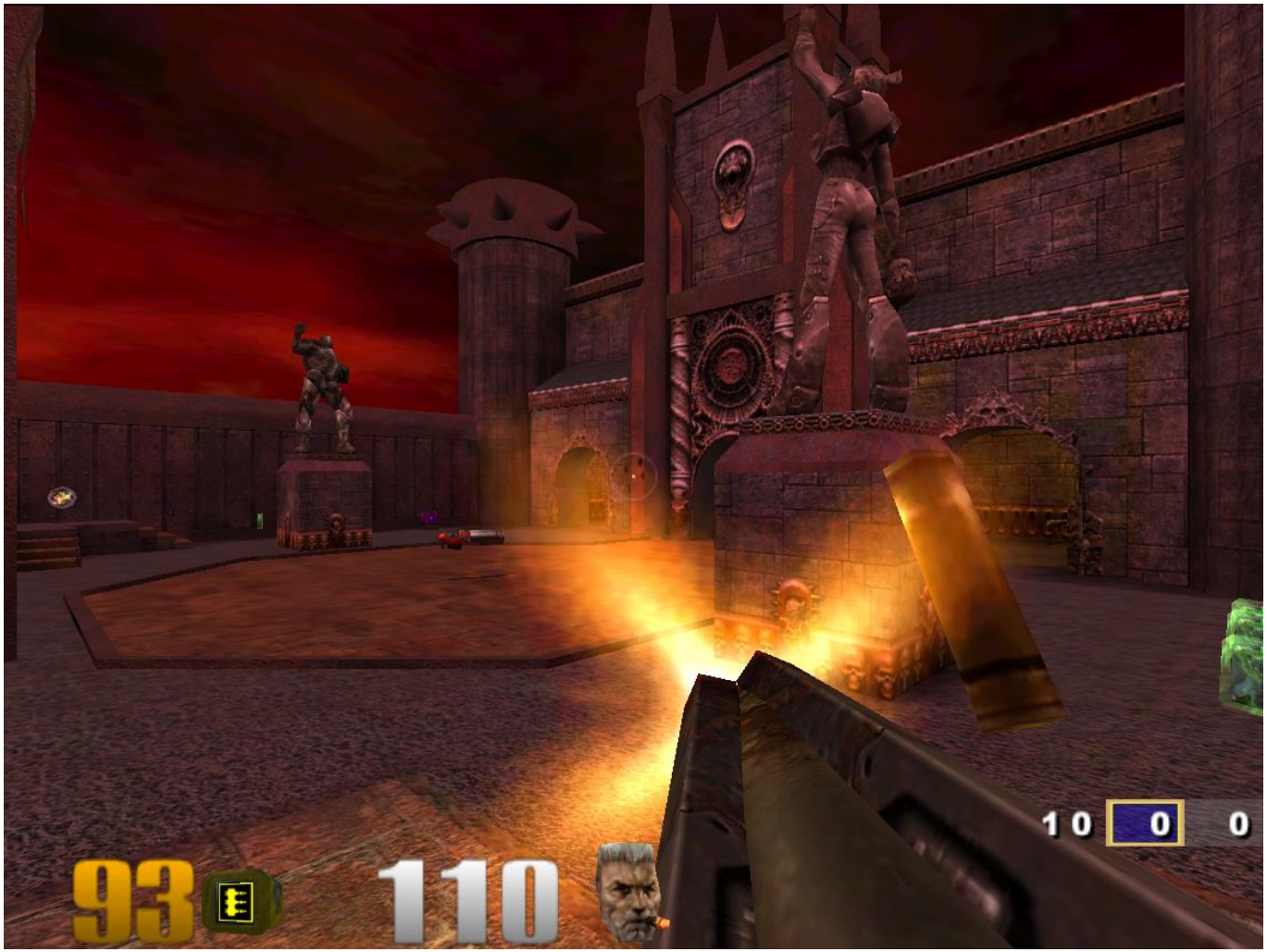
Texture Mapping

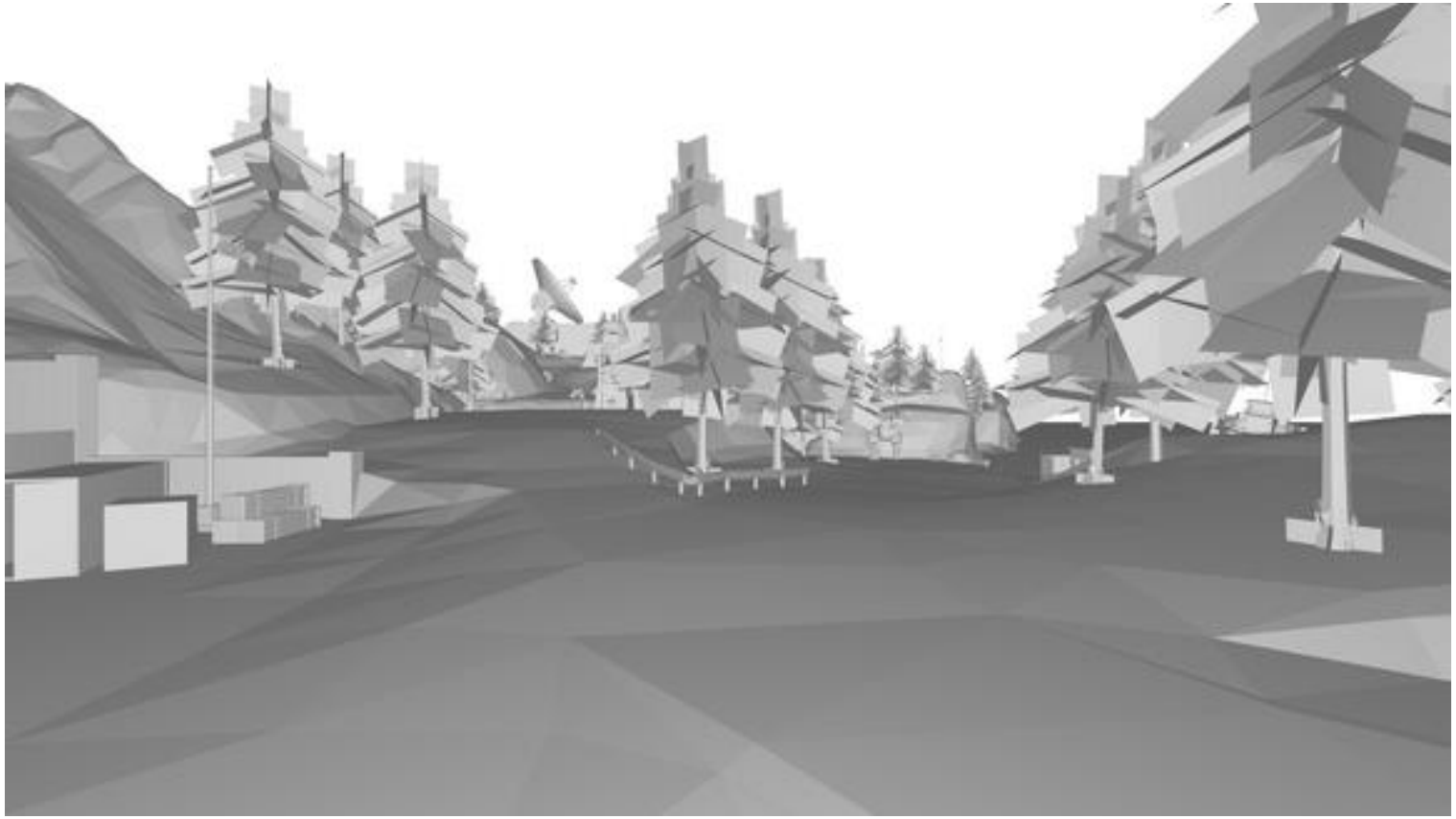
Many slides come from Greg Humphreys, UVA and
Rosalee Wolfe, DePaul tutorial teaching texture mapping visually

Quake 3 scene without textures

Quake 3 scene textured



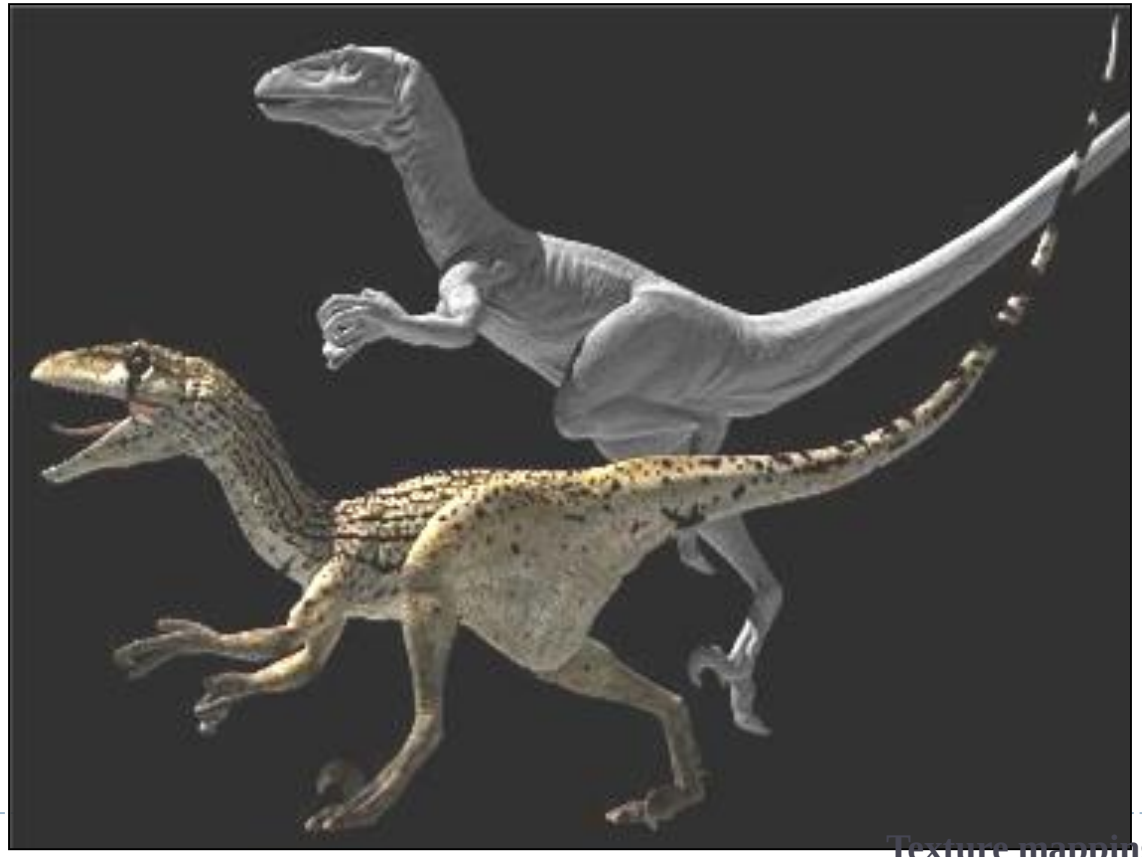
Quake Wars scene without textures



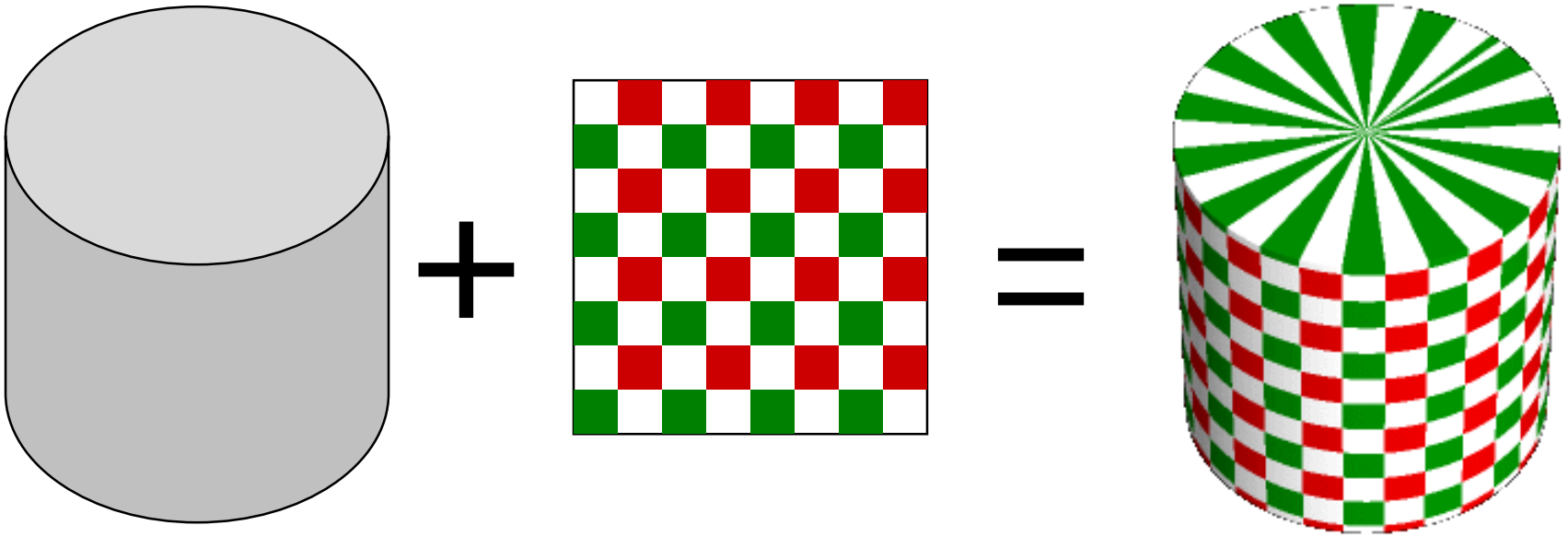
Quake Wars scene with textures and lighting

Adding Visual Detail

- Basic idea: use images instead of more polygons to represent fine scale color variation



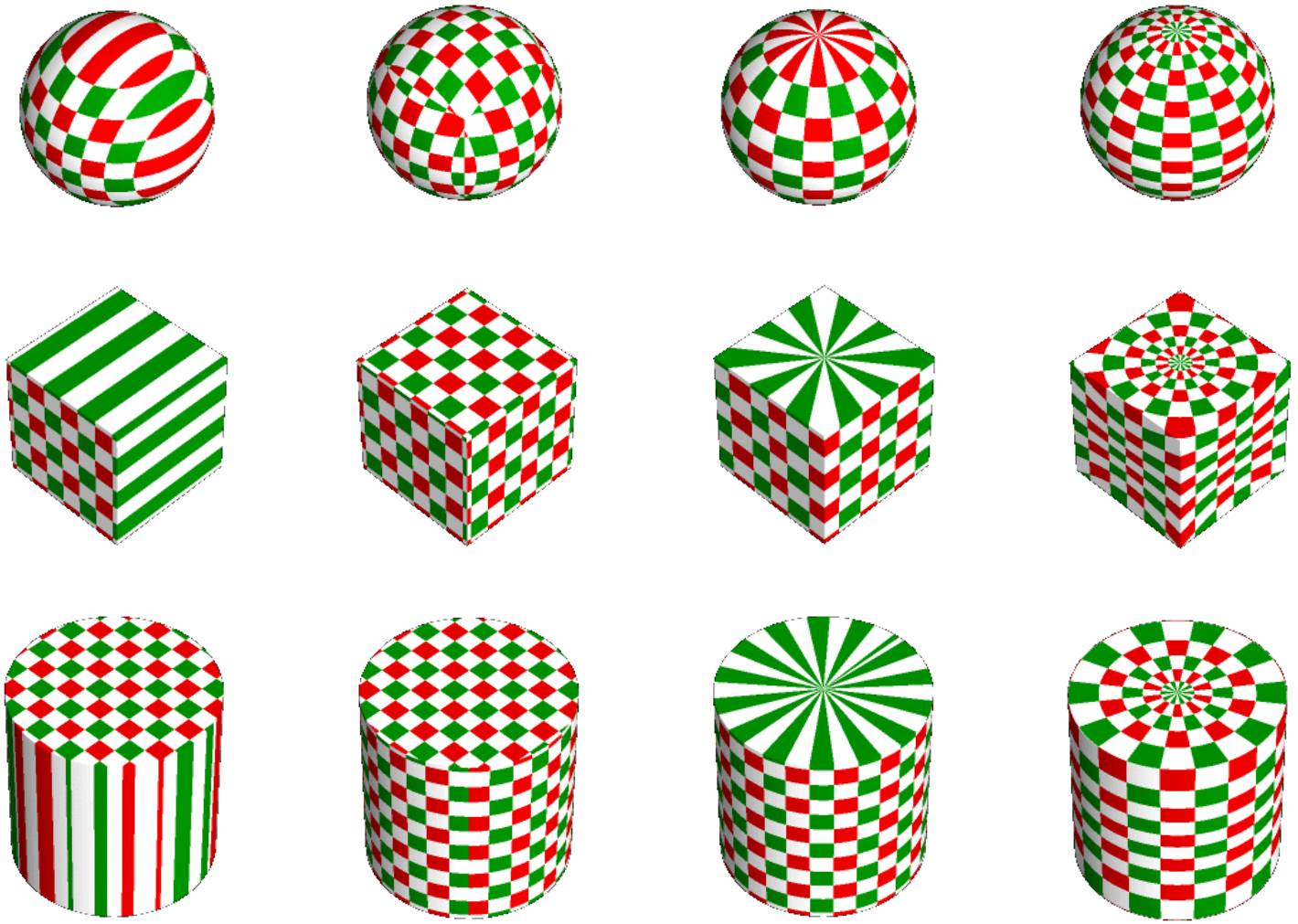
Parameterization



geometry image texture map

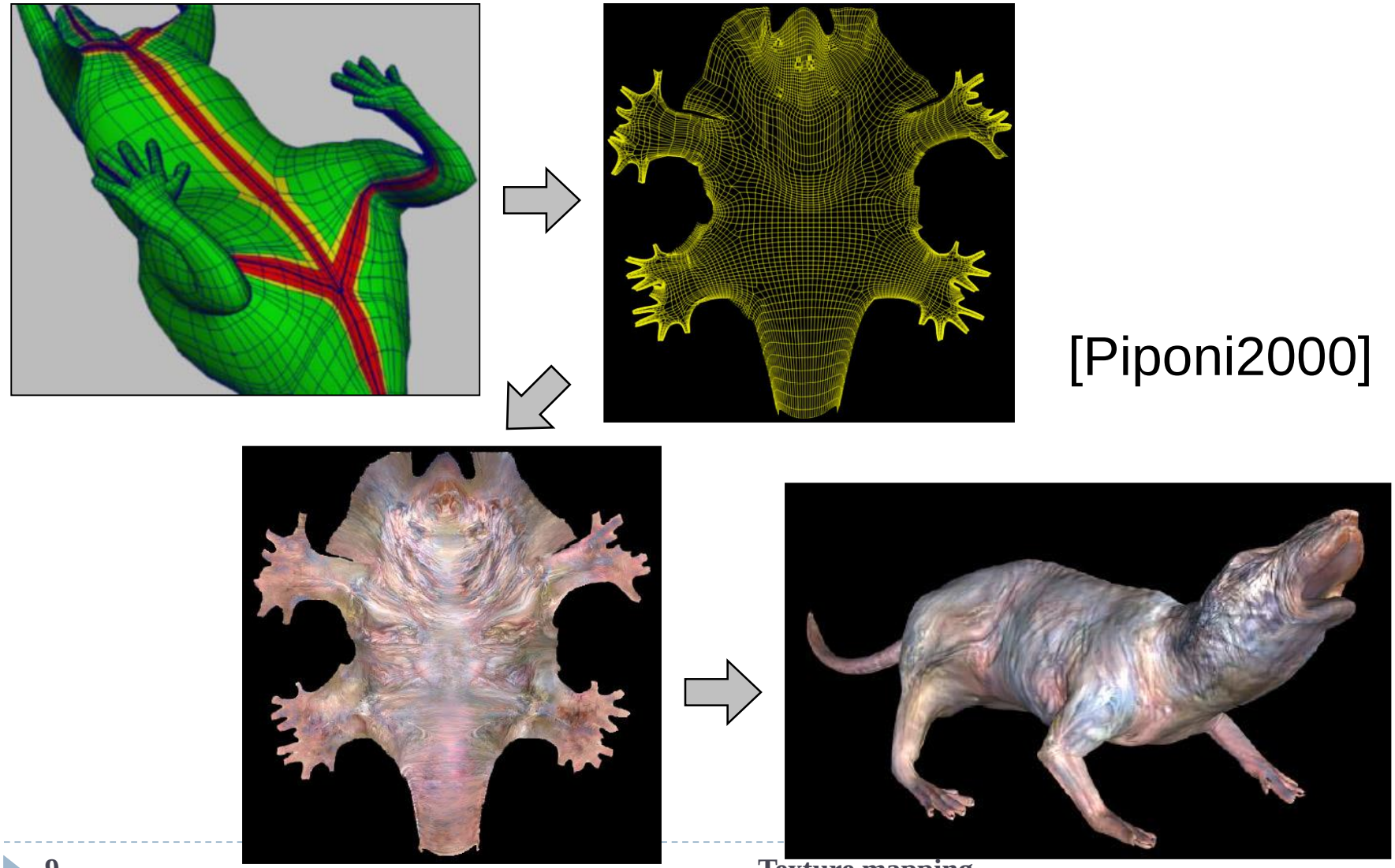
- Q: How do we decide *where* on the geometry each color from the image should go?

Option: Varieties of projections



[Paul Bourke]

Option: unfold the surface



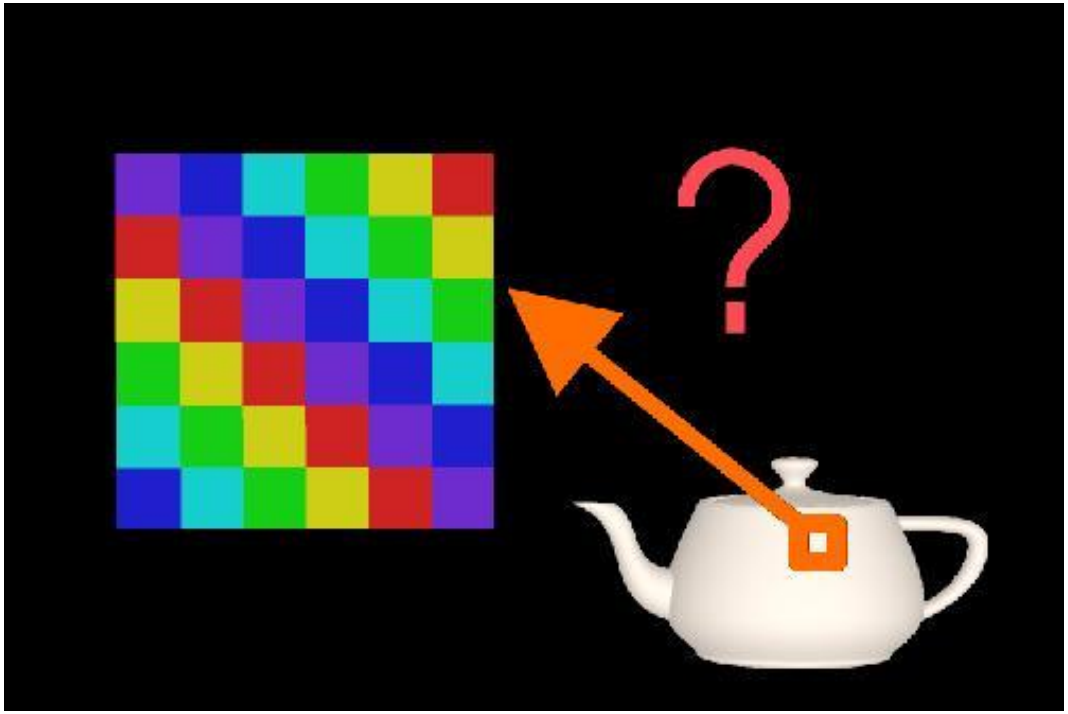
[Piponi2000]

Outline

- Types of projections
- Interpolating texture coordinates
- Broader use of textures

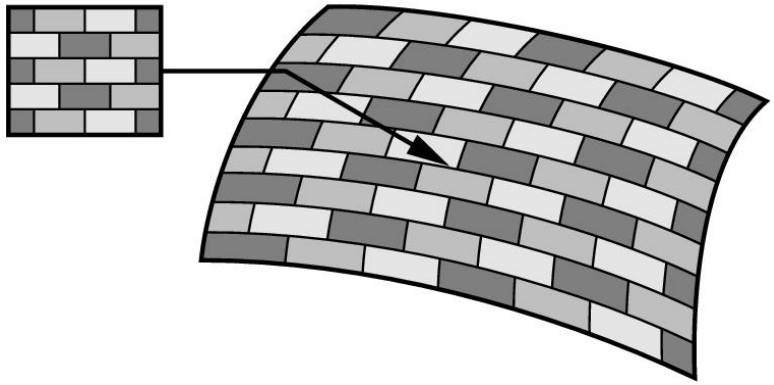
How to map object to texture?

- To each vertex (x,y,z in object coordinates), must associate 2D texture coordinates (s,t)
- So texture fits "nicely" over object



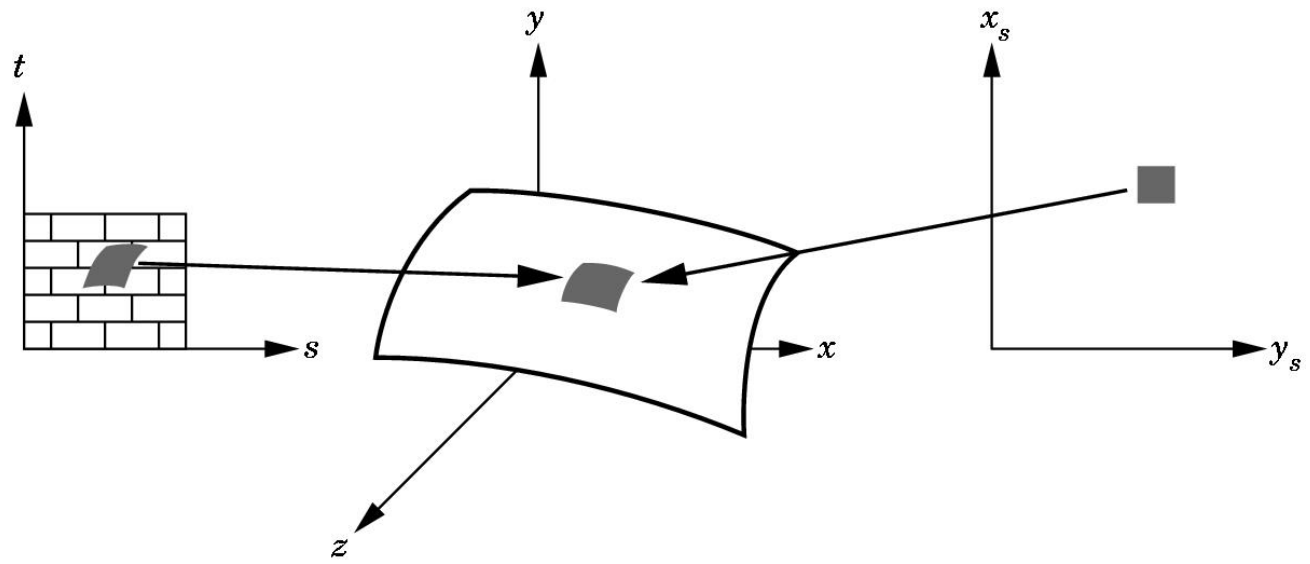
Mapping geometry

- Texture mapping:
  rendering polygons by
  applying images with
  mapped geometry
- (s,t) – *texture coordinates*
- T(s,t) – *texture*
- x(s,t), y(s,t), z(s,t) – *texture map*
  (a mapping of texture onto world coords)
- To render, we really want the *inverse* map: s(x,y,z) and t(x,y,z)



Mapping geometry

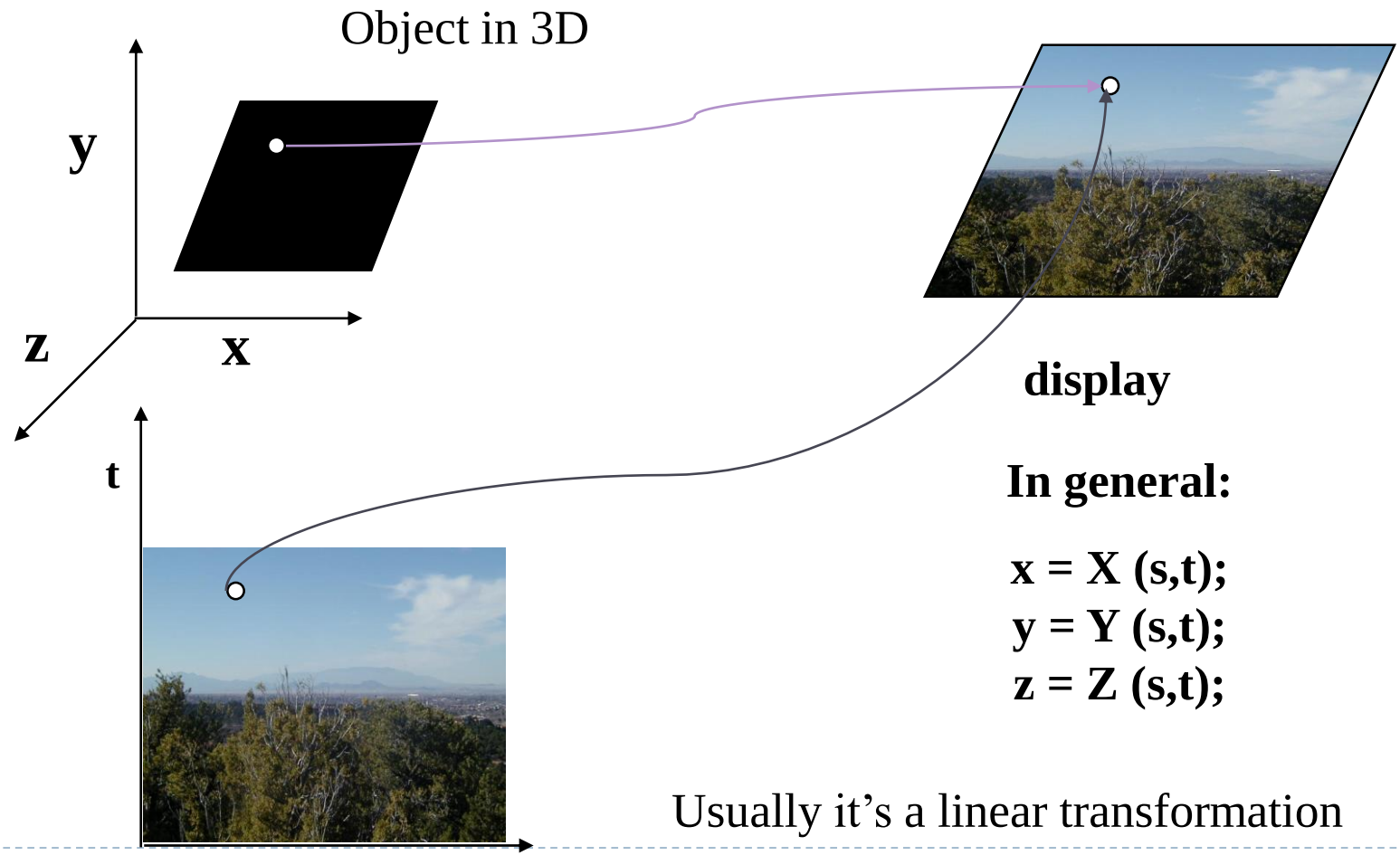
- Also have to map screen coordinates (x_s,y_s) into world coordinates before you can map texture coordinates onto world coordinates



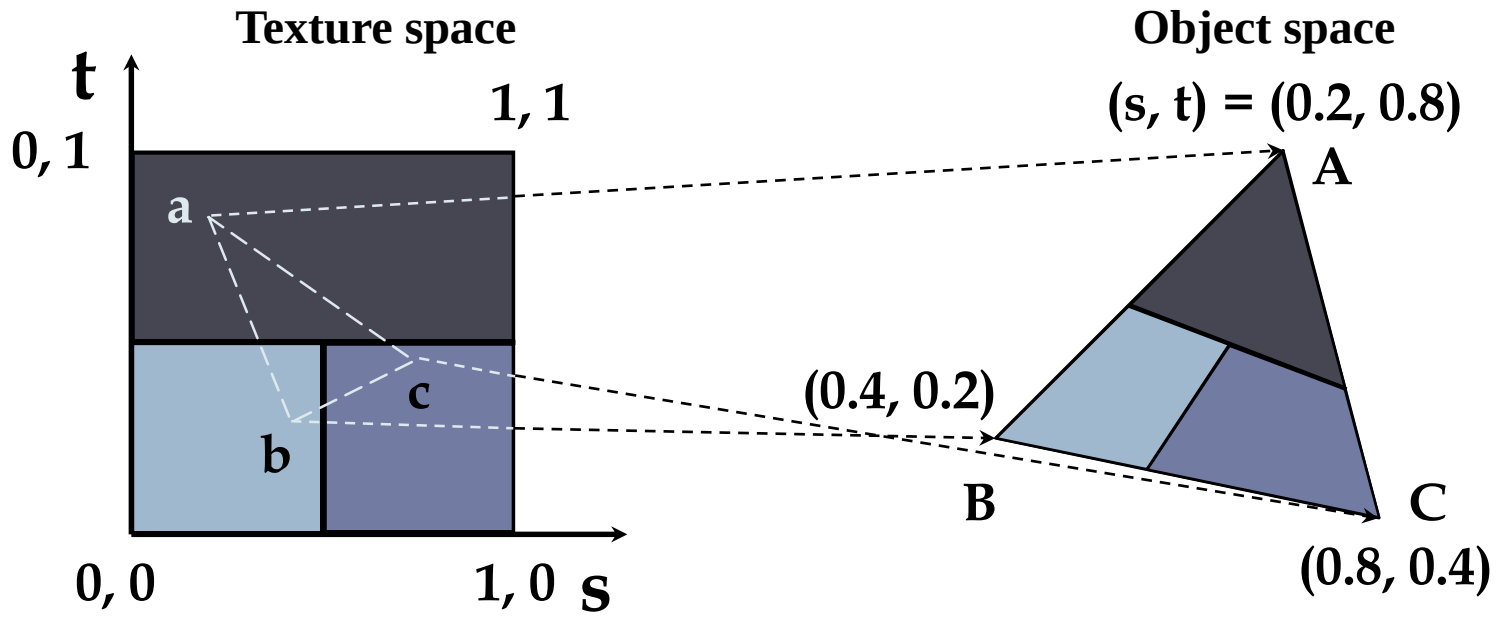
Texture Mapping Coordinates

- Sometimes finding the mapping is easy (rectangle onto rectangle)
- Can use some standard mappings (rectangle onto a sphere, for example)
- Otherwise, it's generally done by hand
  - Automatically "unwrap" shape onto plane
  - Hand-paint or otherwise create custom texture map

Sometimes mapping is easy



Texture coordinates?

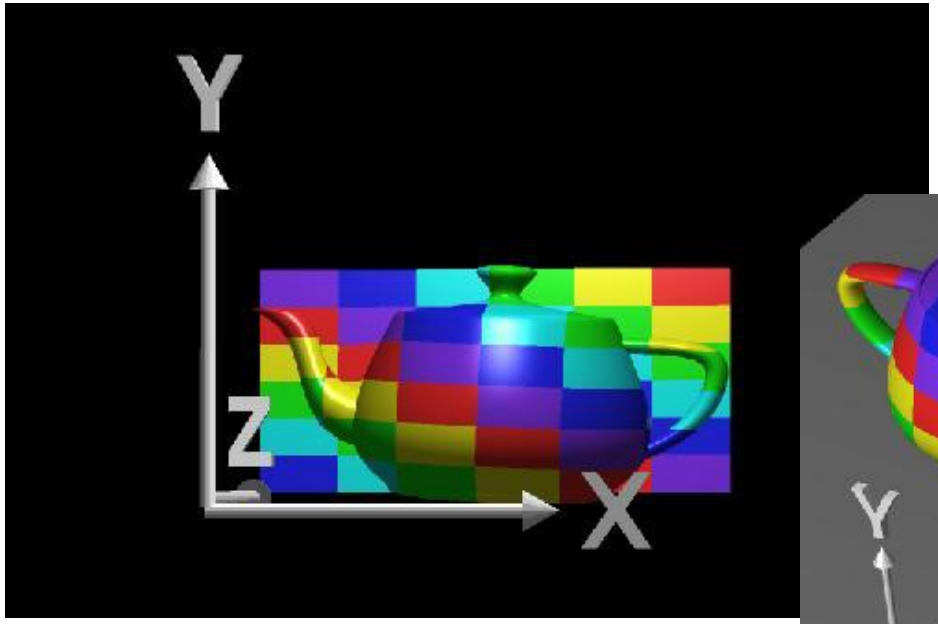


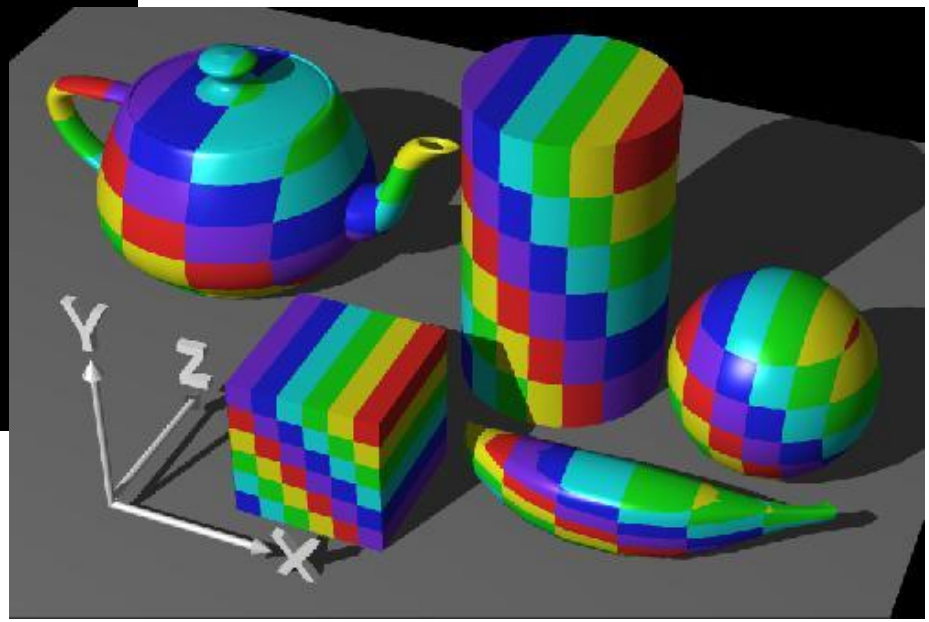
Idea: Use Map Shape

- Map shapes correspond to various projections
  - Planar, Cylindrical, Spherical
- First, map (square) texture to basic map shape
- Then, map basic map shape to object
  - Or vice versa: Object to map shape, map shape to square
- Usually, this is straightforward
  - Maps from square to cylinder, plane, sphere well defined
  - Maps from object to these are simply spherical, cylindrical, cartesian coordinate systems

Planar mapping

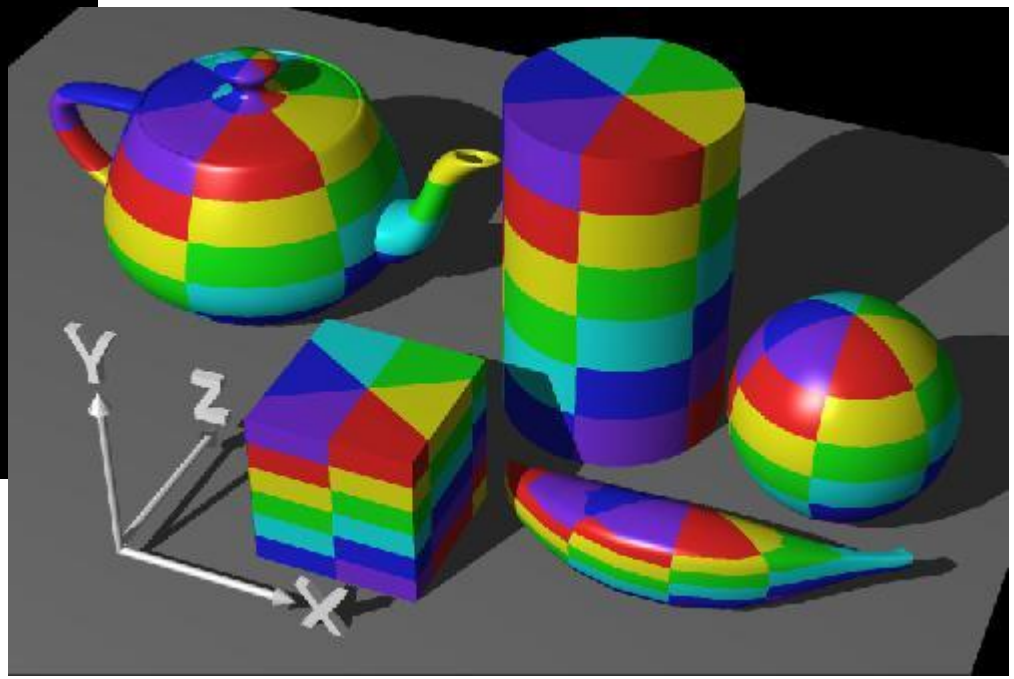
- Like projections, drop z coord (s,t) = (x,y)
- Problems: what happens near z = 0?





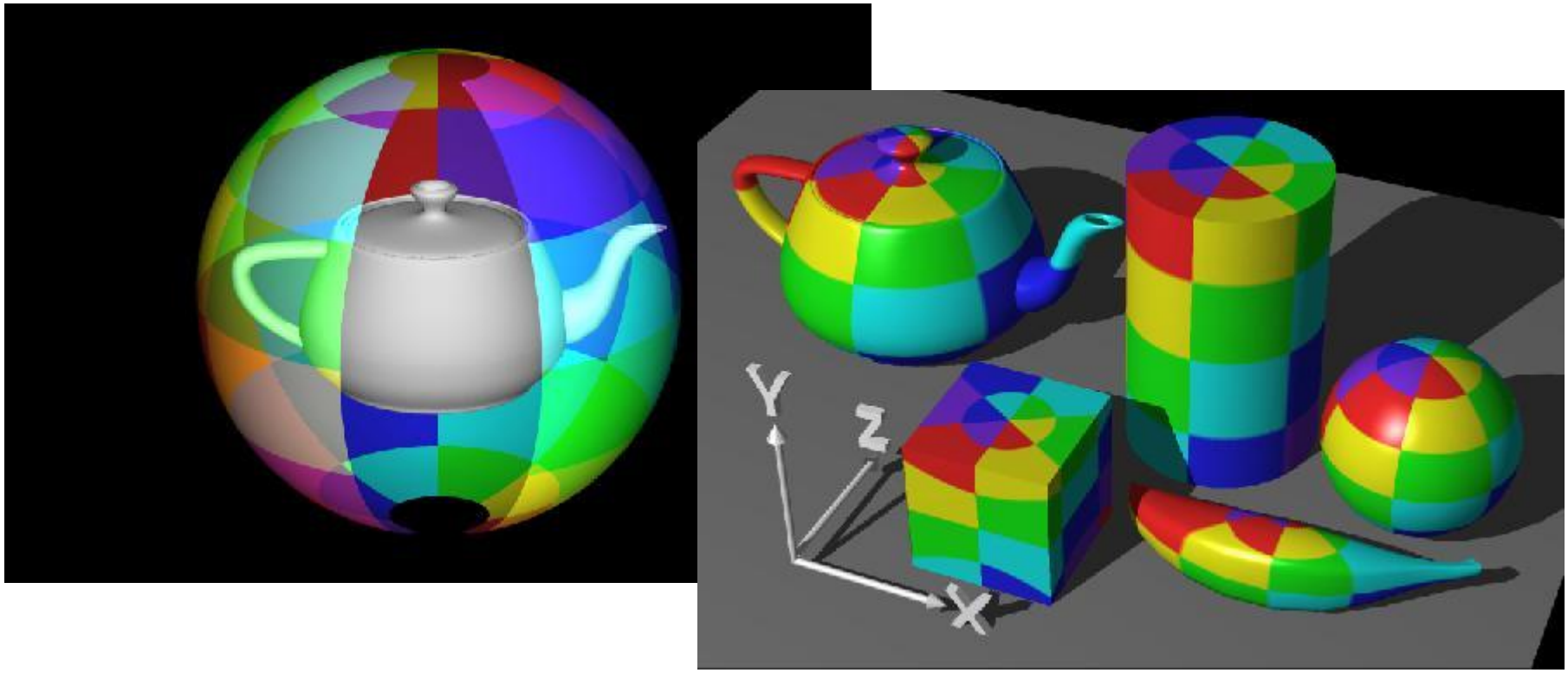
Cylindrical Mapping

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

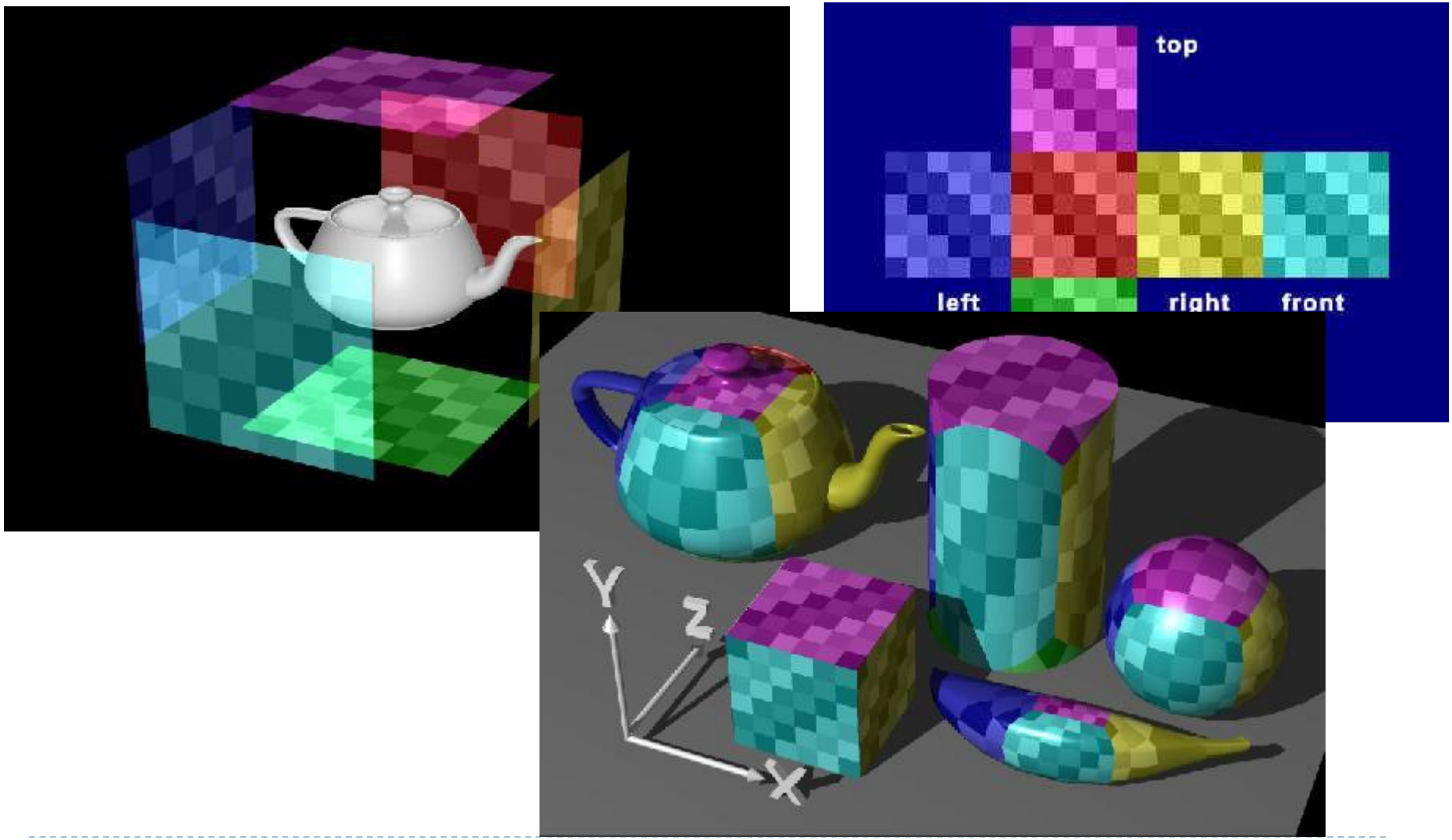


Spherical Mapping

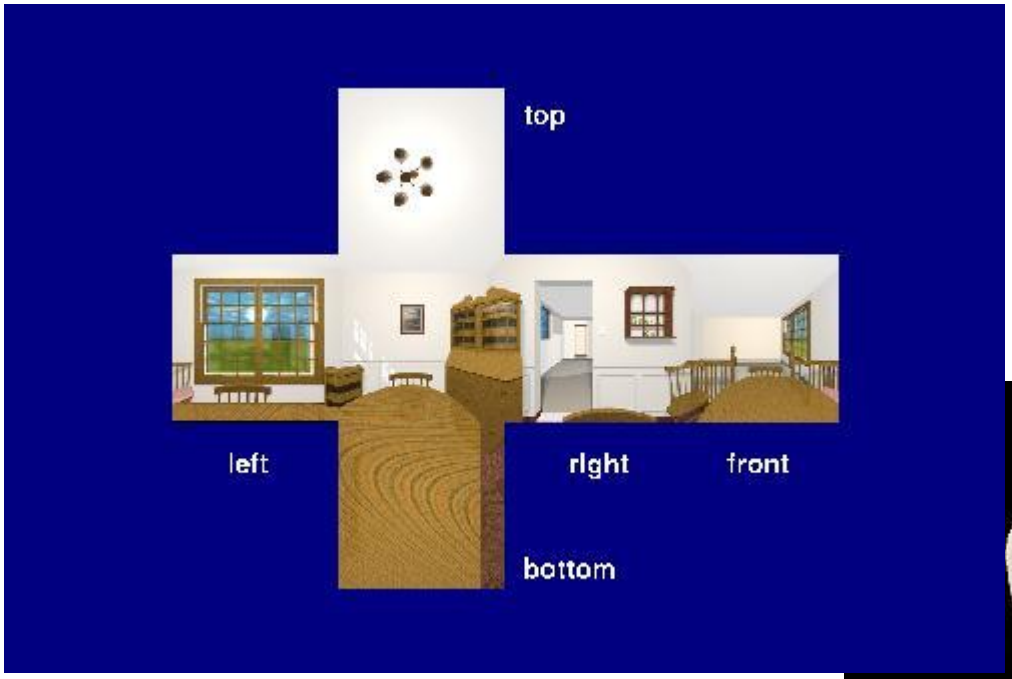
- Convert to spherical coordinates: use latitude/long.
  - Singularities at north and south poles



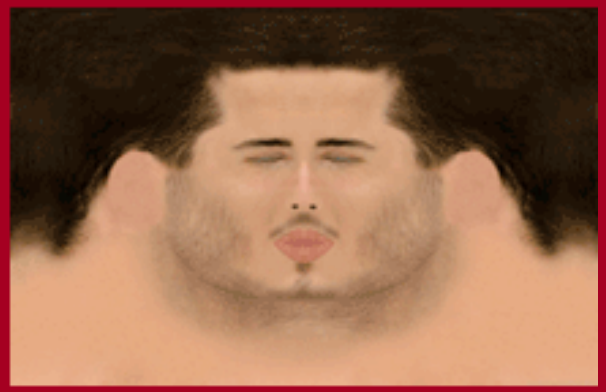
Cube Mapping



Cube Mapping

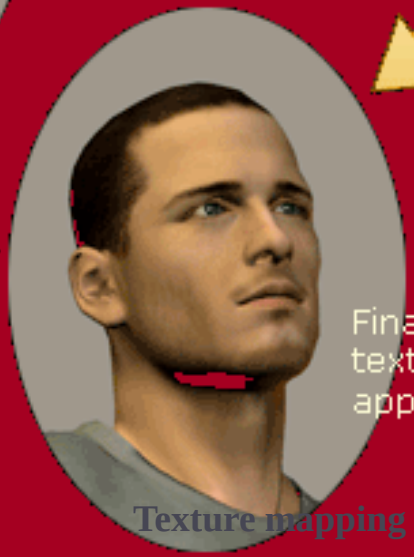


Wireframe



Unwrapped Texture Map

Model with surfaces and lighting applied



Final rendering with texture and lighting applied

Models and texture maps from Poser, a product for creating and rendering human characters.

How to fit textures onto a rectangle

Outline

- Types of projections
- Interpolating texture coordinates
- Broader use of textures

Mapping methods

- Imagine modeling a piece of carved wood, or a golf ball, or a spoon, or a tree…
- Rather than modeling the geometry at a very fine level of detail, we may prefer discrete methods:
  - texture mapping
  - bump mapping
  - normal mapping
  - parallax mapping

Texture mapping

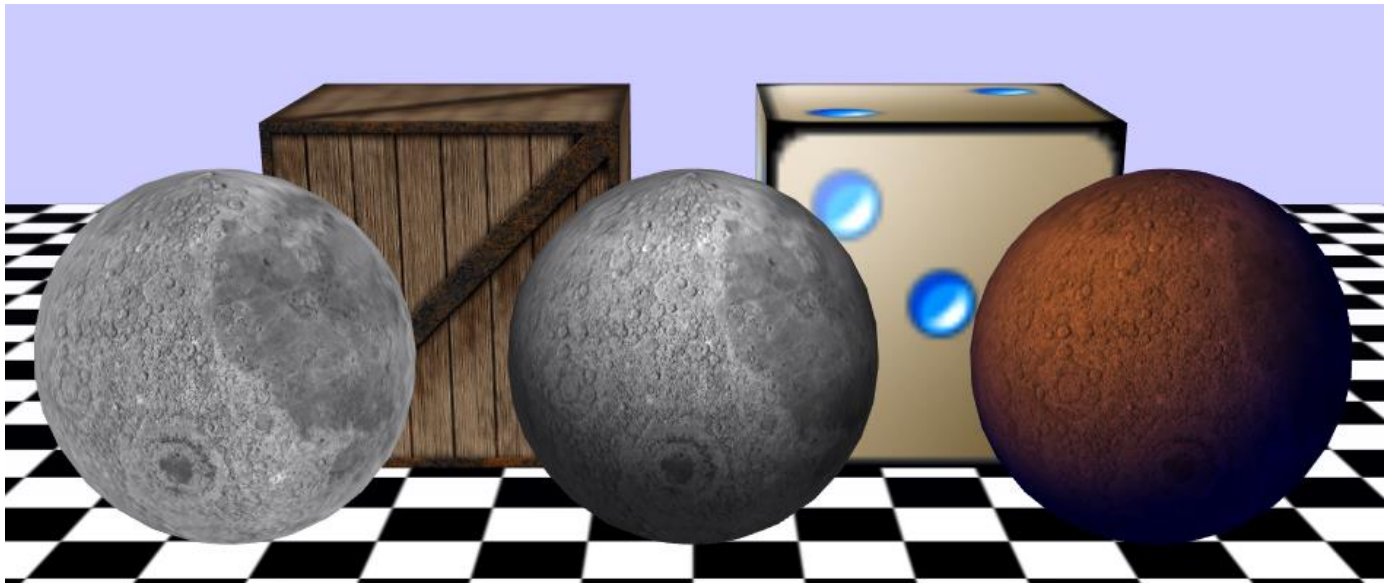
- Texture mapping: applying images to rendered polygons.
- A separate *texture buffer* on the graphics card stores texture images
  - Texture "pixels" are called *texels*
  - Textures may be 1-, 2-, or 3- (or 4-!) dimensional (mapped onto 1-, 2-, or 3-dimensional surfaces)
  - The need for texture images is why you have 1GB graphics cards (and repetitive patterns!)

Texture Mapping in Three.js

Texture Mapping in Three.js

- Specify texture map in material object parameters

```
var moonTex = THREE.ImageUtils.loadTexture( 'images/moon.jpg' );
var moonMaterial = new THREE.MeshLambertMaterial({ map: moonTex });
```



Texture Mapping with Three.js cont.

- Starting from r87 of the library we have another texture loader, with slightly larger capabilities :

```
var moonTex = THREE.TextureLoader().load('images/moon.jpg');
var moonMat = new THREE.MeshLambertMaterial({ map: moonTex });
```

- Details in threejs.org documentation

Texture Object

- Object loaded by TextureLoader().load can be modfied:
- Constructor:

```
Texture( image, mapping, wrapS,
wrapT, magFilter, minFilter,
format, type, anisotropy )
```

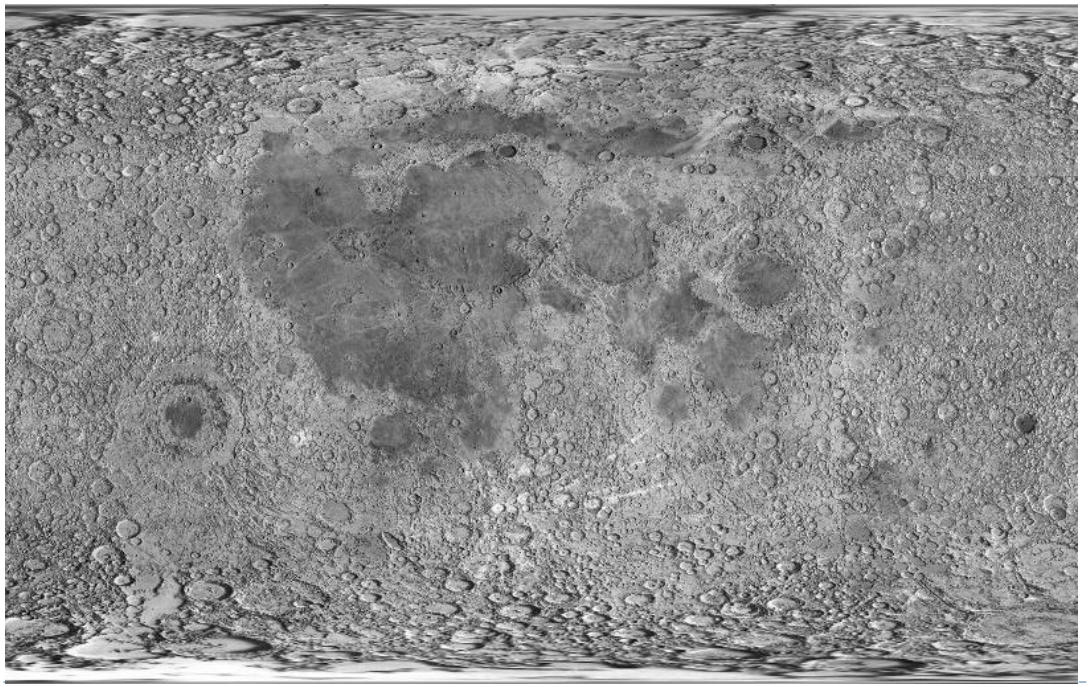
Basic Texture Mapping

```
//Cubes
//  Note: when using a single image, it will appear on each of the faces.
//  Six different images (one per face) may be used if desired.

var cubeGeometry = new THREE.CubeGeometry( 85, 85, 85 );
var crateTexture = new THREE.ImageUtils.loadTexture( 'images/crate.gif');
var crateMaterial = new THREE.MeshBasicMaterial( { map: crateTexture } );
var crate = new THREE.Mesh( cubeGeometry.clone(), crateMaterial );
crate.position.set(-60, 50, -100);
scene.add( crate );
```

```
// Cubes
//   Note: when using a single image, it will appear on
each of the faces.
//   Six different images (one per face) may be used if
desired.

var cubeGeometry = new THREE.CubeGeometry( 85, 85, 85 );

var crateTexture = new THREE.ImageUtils.loadTexture(
'images/crate.gif' );
var crateMaterial = new THREE.MeshBasicMaterial( { map:
crateTexture } );
var crate = new THREE.Mesh( cubeGeometry.clone(),
crateMaterial );
crate.position.set(-60, 50, -100);
scene.add( crate );
```

Basic Spherical Texture Mapping

```
// basic moon
var moonTexture = THREE.ImageUtils.loadTexture( 'images/moon.jpg' );
var moonMaterial = new THREE.MeshBasicMaterial( { map: moonTexture } );
var moon = new THREE.Mesh( sphereGeom.clone(), moonMaterial );
moon.position.set(-100, 50, 0);
scene.add( moon );
```



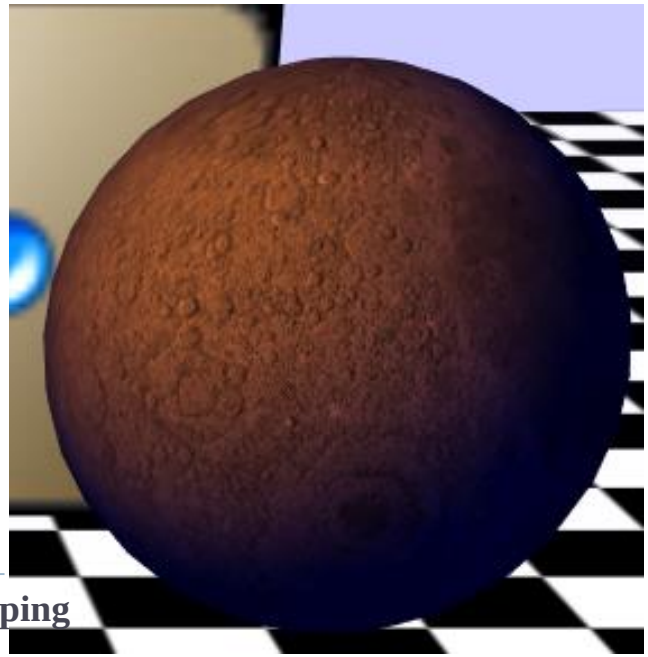
Lighting and Shading

```
// shaded moon -- side away from light picks up AmbientLight's color.
var moonTexture = THREE.ImageUtils.loadTexture( 'images/moon.jpg' );
var moonMaterial = new THREE.MeshLambertMaterial( {map: moonTexture} );
var moon = new THREE.Mesh( sphereGeom.clone(), moonMaterial );
moon.position.set(0, 50, 0);
scene.add( moon );
```

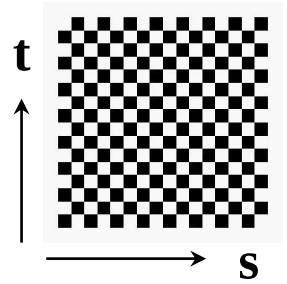
Texture and Color

```
MeshLambertMaterial({ map: moonTexture, color:0xff8800;
ambient: 0x0000ff } ) ;
```

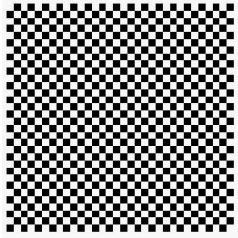
```
// colored moon
var moonTexture = THREE.ImageUtils.loadTexture( 'images/moon.jpg' );
var moonMaterial = new THREE.MeshLambertMaterial( {map: moonTexture,
color: 0xff8800, ambient: 0x0000ff} );
var moon = new THREE.Mesh( sphereGeom.clone(), moonMaterial );
moon.position.set(100, 50, 0);
scene.add( moon );
```



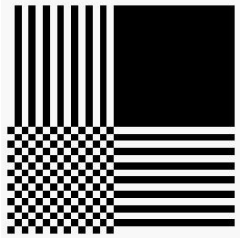
Texture filtering: Repeat and Clamp



Texture



REPEAT

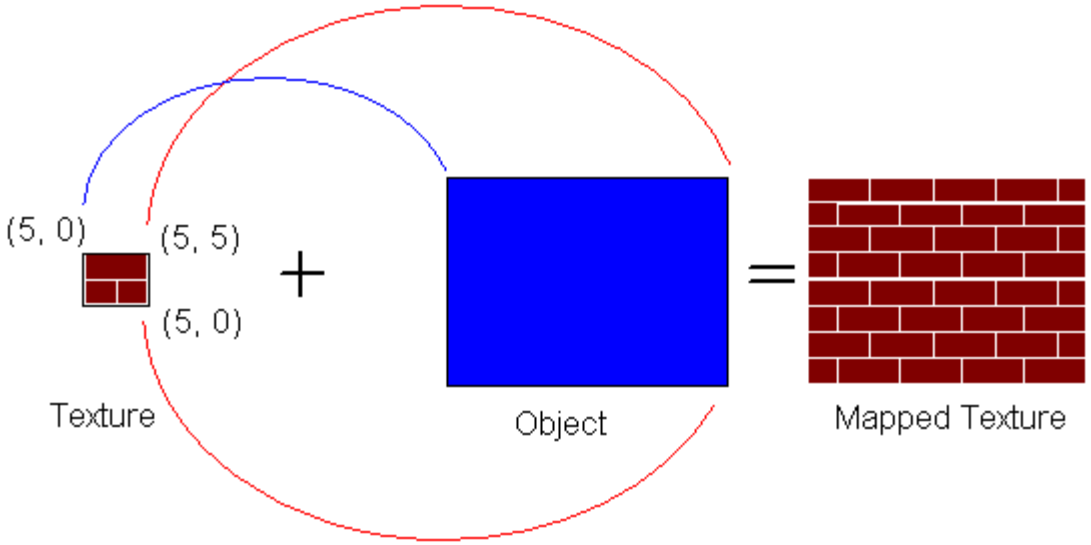


CLAMP

```
var texture = THREE.ImageUtils.loadTexture('checker.jpg');
texture.wrapS = THREE.ClampToEdgeWrapping;
texture.wrapT = THREE.ClampToEdgeWrapping;
texture.repeat.set( 4, 4 );
```

Texture filtering: Repeat and Clamp

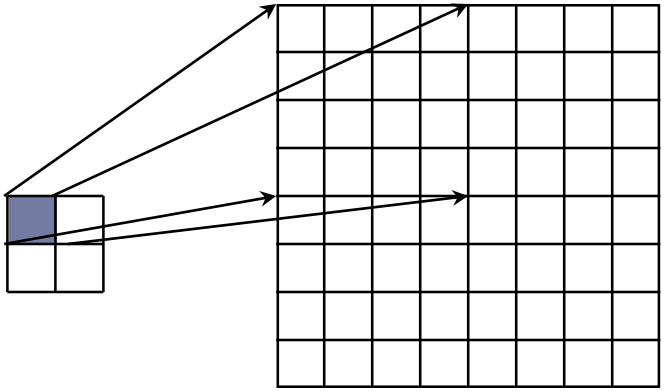
Repeat mode is much more common




```
var texture = THREE.ImageUtils.loadTexture('bricks.jpg');
texture.wrapS = THREE.RepeatWrapping;
texture.wrapT = THREE.RepeatWrapping;
texture.repeat.set( 4, 4 );
```

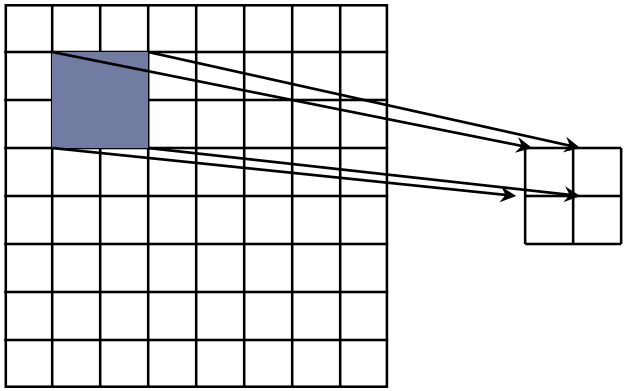
Texture filtering: magnification and minification

We have to set an interpolation algorithm for textures when magnifying and minifying objects



Texture **Polygon pixels**

Magnification – one texel mapped onto many pixels



Texture **Polygon pixels**

Minification many texels compressed into one pixel

Texture filtering

Filtrowanie określa w jaki sposób tekstura będzie nakładana na obiekt oraz jak będzie się zmieniać przy zmianie położenia obiektu względem obserwatora.

```
var texture = THREE.ImageUtils.loadTexture('bricks.jpg');
texture.magFilter = THREE.LinearFilter;
texture.minFilter = THREE.LinearMipMapLinearFilter;
texture.repeat.set( 4, 4 );
```

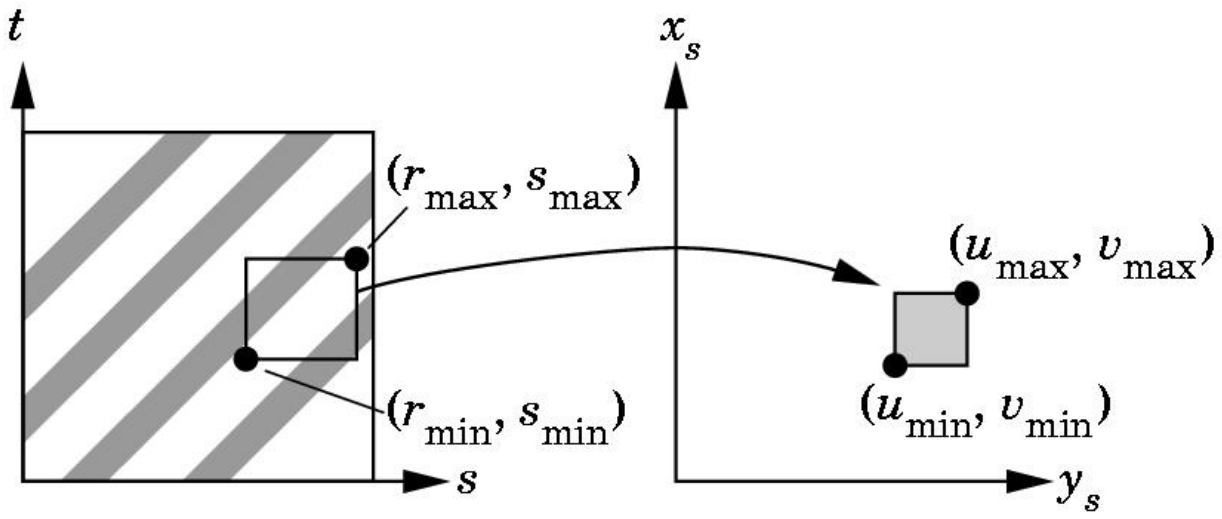
Dla powiększenia mamy: `NearestFilter` i `LinearFilter`
Dla pomniejszenia – wszystko:

Filtrowanie tekstur: wartości parametrów w three.js

Wartość	
THREE.NearestFilter	
THREE.LinearFilter	
THREE.NearestMipMapNearestFilter	
THREE.NearestMipMapLinearFilter	
THREE.LinearMipMapNearestFilter	
THREE.LinearMipMapLinearFilter	

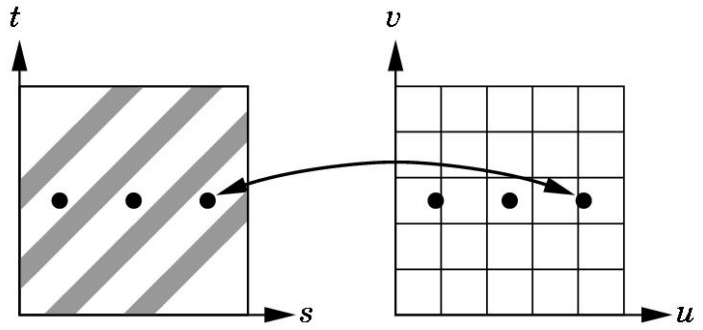
Pixel color from texture map?

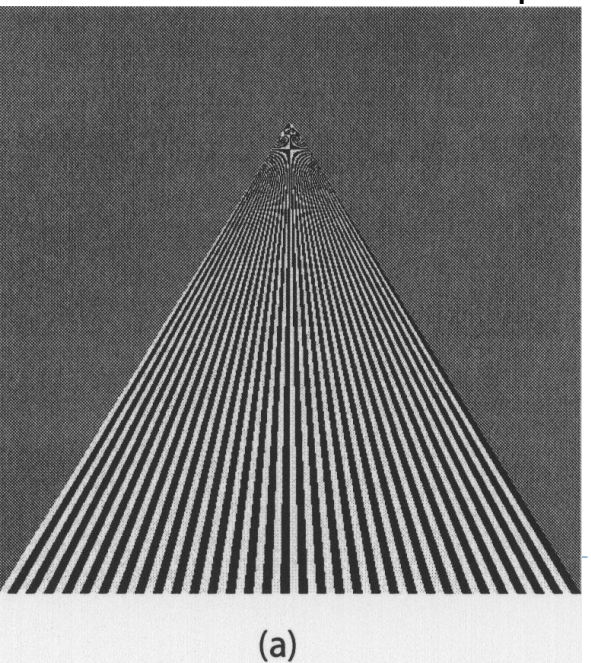
- How do we get the pixel's color from the texture map?
  - We could map the center of the pixel to the texture map and use the color we find there

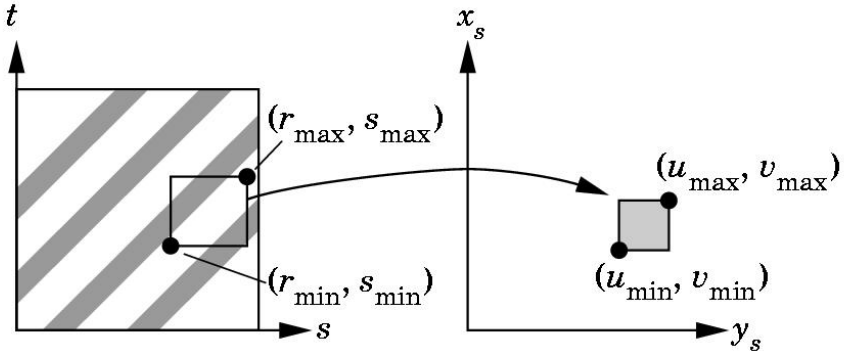


Pixel color from texture map?

- How do we get the pixel's color from the texture map?
  - We could map the center of the pixel to the texture map and use the color we find
    - But this can result in severe aliasing and flickering
    - Better to map the whole pixel and average. But won't that be







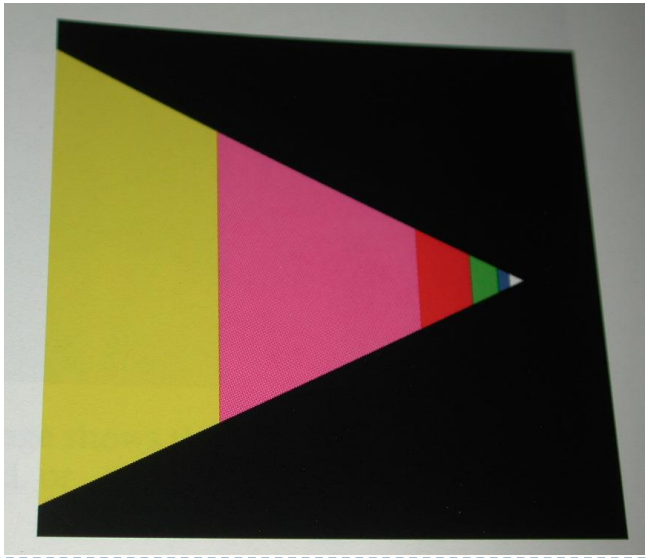
Aliasing and Filtering

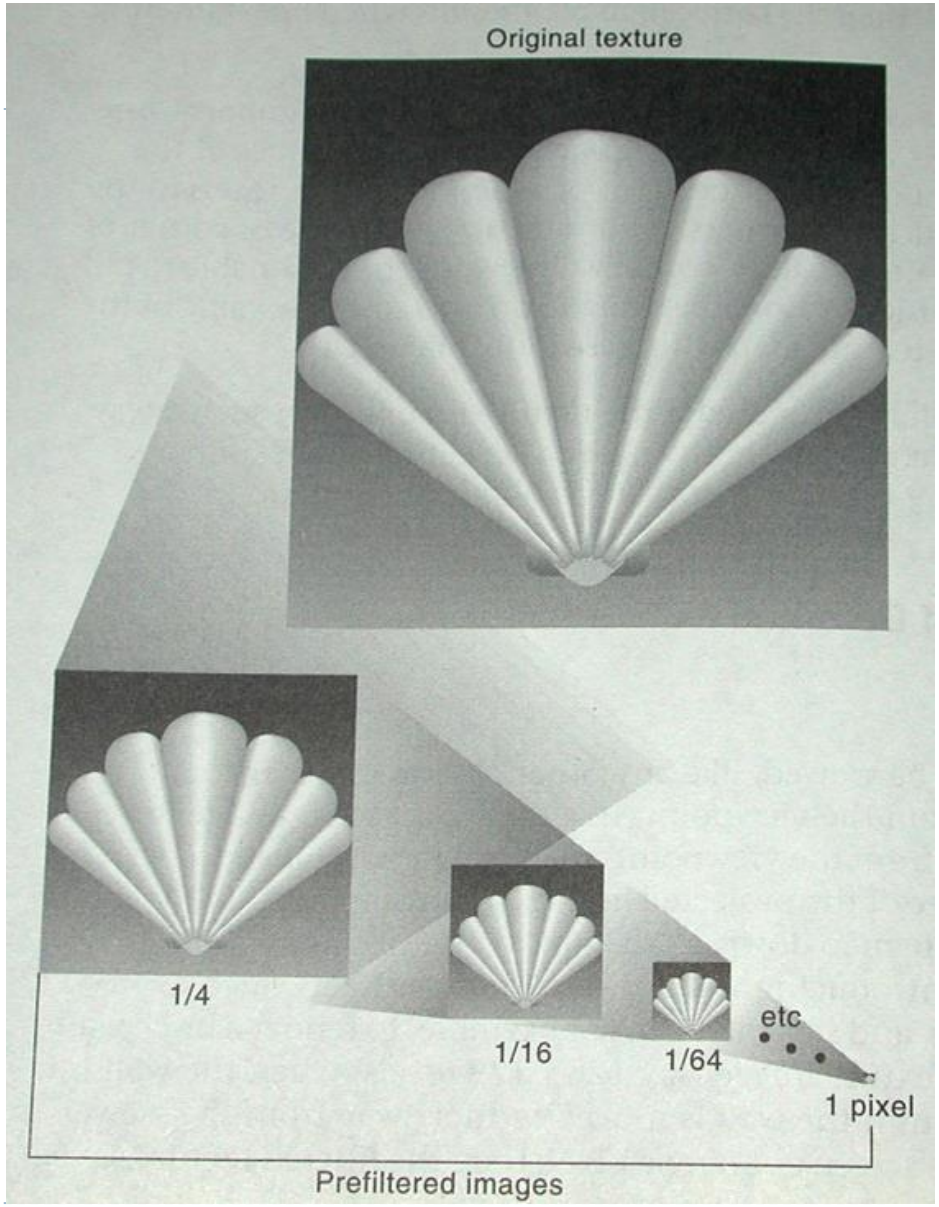
- What happens if texture is too small or too big?
- What happens when you zoom way in or out on a textured object?

- (Moire patterns)

Mipmaps

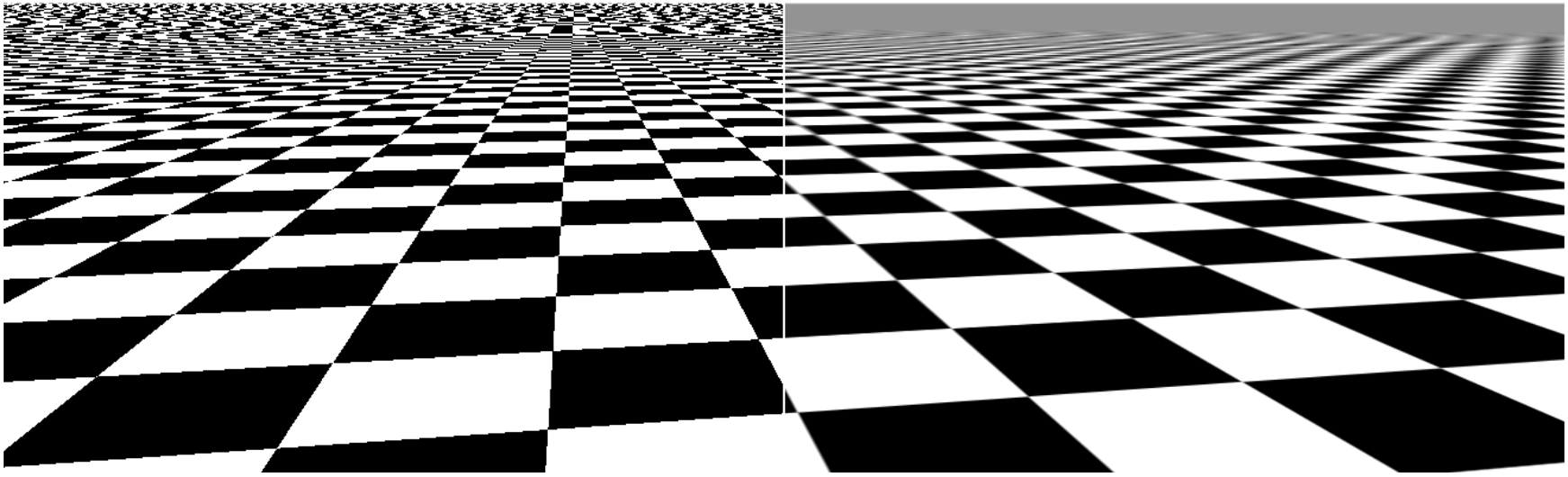
- Make pre-filtered images of size 1x1, 2x2, 4x4, …, nxn
- Choose the best
- Examples: [1][2]





Mipmapping example

- http://www.ecole.ensicaen.fr/~simonl/files/OpenGL/Cours/Lesson3/WebGL/mipmaping/webgl_materials_texture_mipmap.html



Code fragment from previous image

```
var maxAnisotropy = renderer.getMaxAnisotropy();

var texture1 = THREE.ImageUtils.loadTexture
                ( "textures/checkerboard.jpg" );
var material1 = new THREE.MeshPhongMaterial(
                { color: 0xffffff, map: texture1 } );
texture1.generateMipmaps = false;
texture1.minFilter = texture1.magFilter = THREE.NearestFilter;
texture1.wrapS = texture1.wrapT = THREE.RepeatWrapping;
texture1.repeat.set( 256, 256 );

var texture2 = THREE.ImageUtils.loadTexture
               ( "textures/checkerboard.jpg" );
var material2 = new THREE.MeshPhongMaterial
               ( { color: 0xffffff, map: texture2 } );
texture2.generateMipmaps = true;
texture2.minFilter = THREE.LinearMipMapLinearFilter;
texture2.magFilter = THREE.LinearFilter;
texture2.wrapS = texture2.wrapT = THREE.RepeatWrapping;
texture2.repeat.set( 256, 256 );
```

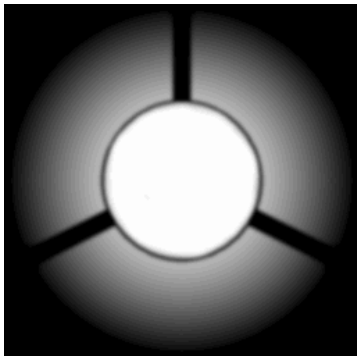
Mipmaps in Three.js

```
E.Texture = function ( image, mapping, wrapS, wrapT, magFilter, minFilter, format, type, anisotropy ) {

    THREE.TextureLibrary.push( this );

    this.id = THREE.TextureIdCount ++;

    this.image = image;

    this.mapping = mapping !== undefined ? mapping : new THREE.UVMapping();

    this.wrapS = wrapS !== undefined ? wrapS : THREE.ClampToEdgeWrapping;
    this.wrapT = wrapT !== undefined ? wrapT : THREE.ClampToEdgeWrapping;

    this.magFilter = magFilter !== undefined ? magFilter : THREE.LinearFilter;
    this.minFilter = minFilter !== undefined ? minFilter : THREE.LinearMipMapLinearFilter;

    this.anisotropy = anisotropy !== undefined ? anisotropy : 1;

    this.format = format !== undefined ? format : THREE.RGBAFormat;
    this.type = type !== undefined ? type : THREE.UnsignedByteType;

    this.offset = new THREE.Vector2( 0, 0 );
    this.repeat = new THREE.Vector2( 1, 1 );
```

Handling Multiple Textures

- Using multiple texture objects:
  - Make two texture samplers
  - Read the values from both
  - Combine however you like in fragment shader
- Fragment shader example:

```
baseColor  = texture2D(s_baseMap, v_texCoord);
lightColor = texture2D(s_lightMap, v_texCoord);
gl_FragColor = baseColor * (lightColor + 0.25);
```


Texture mapping

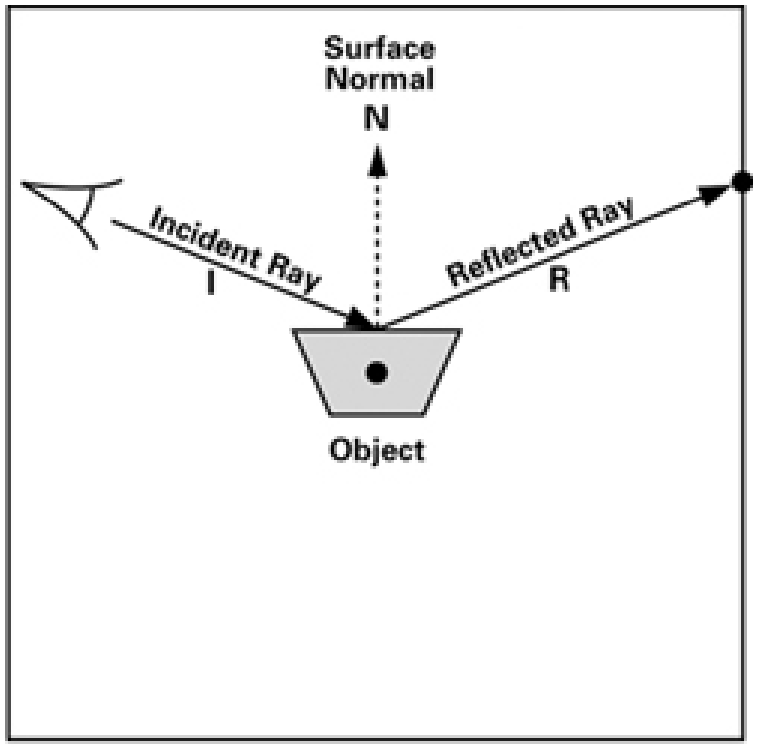
Three.js Multiple Textures

```
var materials = [];
for (var i=0; i<6; i++) {
  var img = new Image();
  img.src = i + '.png';
  var tex = new THREE.Texture(img);
  img.tex = tex;
  img.onload = function() {
    this.tex.needsUpdate = true;
  };
  var mat = new THREE.MeshBasicMaterial({color: 0xffffff, map: tex});
  materials.push(mat);
}
var cubeGeo = new THREE.CubeGeometry(400,400,400,1,1,1, materials);
var cube = new THREE.Mesh(cubeGeo, new THREE.MeshFaceMaterial());
```

Texture mapping

Environment Mapping

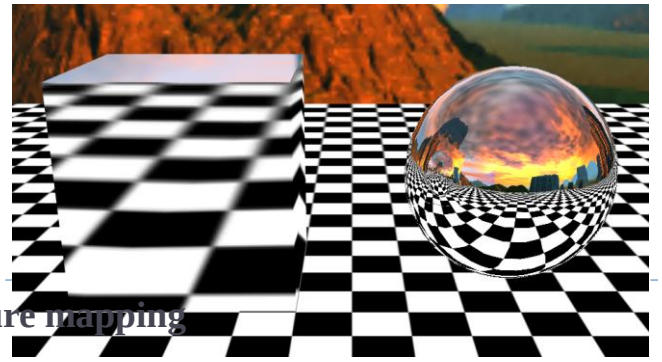
- Compute viewing direction reflection vector
- See where it hits the environment map
- Need:
  - View direction
  - Surface normal at every pt
- Simplifications:
  - Assume environment is distant (only need direction to environment)
  - Use the same environment map for all objects (no one will notice for curved surfaces)



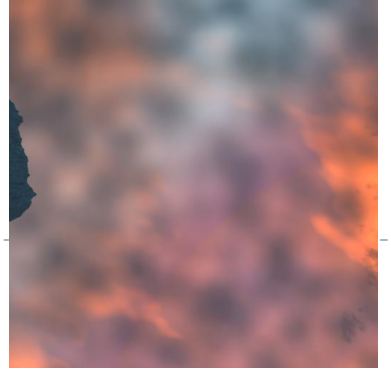
Env Mapping in Three.js

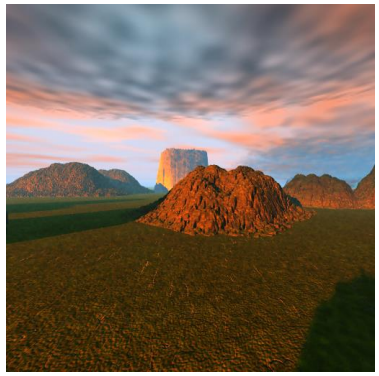
```
mirrorCubeCamera = new Three.CubeCamera(0.1,5000,512);
THREE.MeshBasicMaterial( { envMap:
    mirrorCubeCamera.renderTarget } );
```

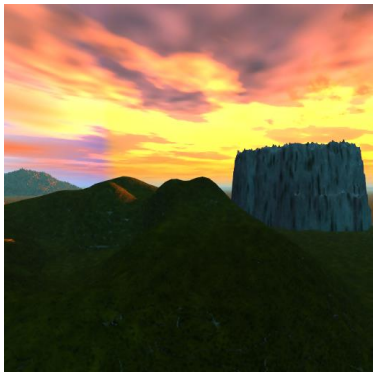
```
var cubeGeom = new THREE.CubeGeometry(100, 100, 100, 1, 1, 1);
mirrorCubeCamera = new THREE.CubeCamera( 0.1, 5000, 512 );
// mirrorCubeCamera.renderTarget.minFilter = THREE.LinearMipMapLinearFilter;
scene.add( mirrorCubeCamera );
var mirrorCubeMaterial = new THREE.MeshBasicMaterial( { envMap: mirrorCubeCamera.renderTarget } );
mirrorCube = new THREE.Mesh( cubeGeom, mirrorCubeMaterial );
mirrorCube.position.set(-75,50,0);
mirrorCubeCamera.position = mirrorCube.position;
scene.add(mirrorCube);

var sphereGeom =  new THREE.SphereGeometry( 50, 32, 16 ); // radius, segmentsWidth, segmentsHeight
mirrorSphereCamera = new THREE.CubeCamera( 0.1, 5000, 512 );
// mirrorCubeCamera.renderTarget.minFilter = THREE.LinearMipMapLinearFilter;
scene.add( mirrorSphereCamera );
var mirrorSphereMaterial = new THREE.MeshBasicMaterial( { envMap: mirrorSphereCamera.renderTarget } );
mirrorSphere = new THREE.Mesh( sphereGeom, mirrorSphereMaterial );
mirrorSphere.position.set(75,50,0);
mirrorSphereCamera.position = mirrorSphere.position;
scene.add(mirrorSphere);
```

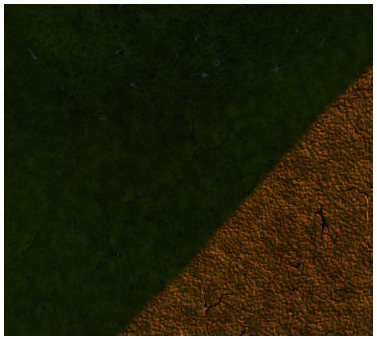


Skybox







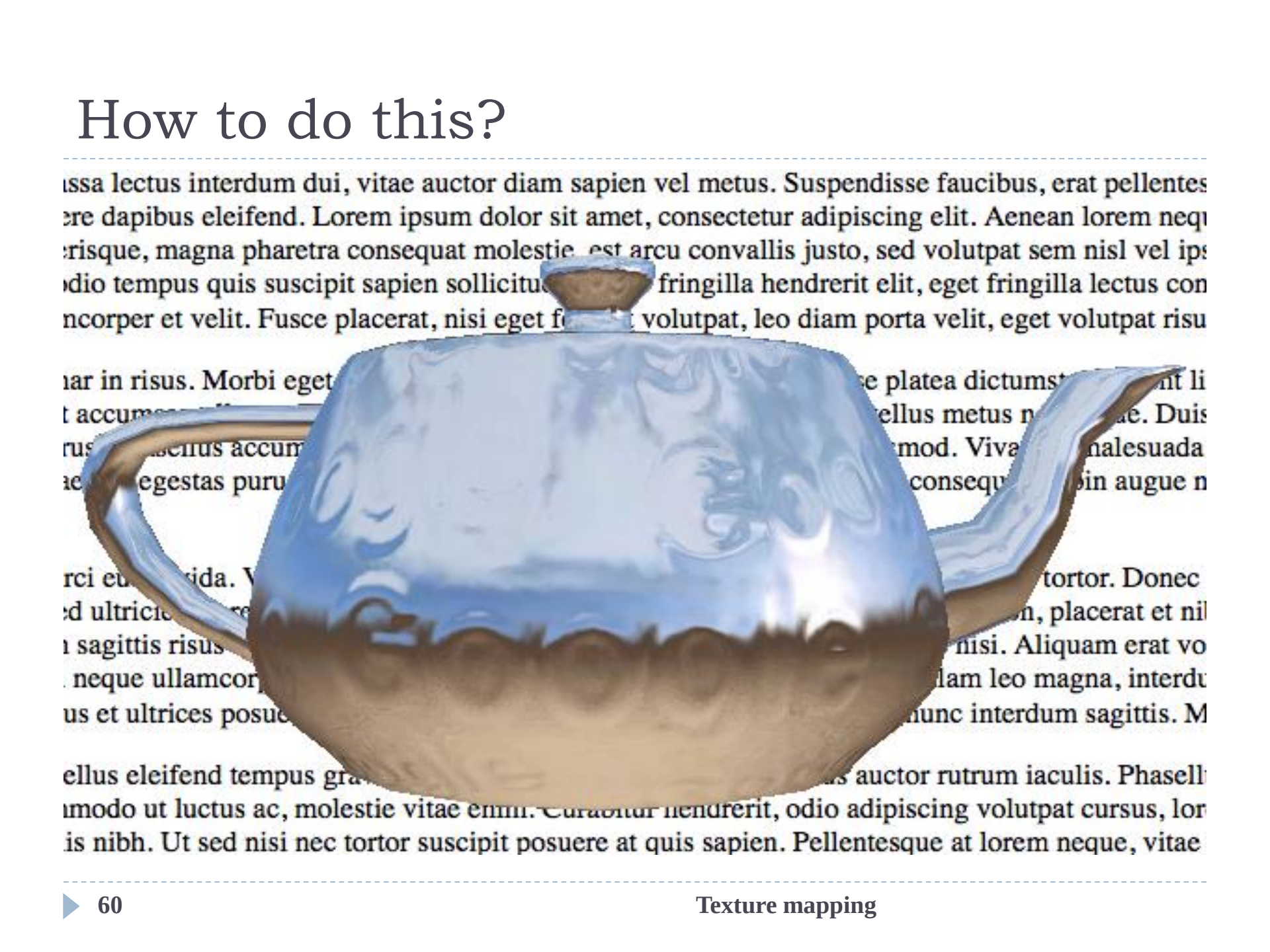


```
var materialArray = [];
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-xpos.png' ) }));
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-xneg.png' ) }));
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-ypos.png' ) }));
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-yneg.png' ) }));
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-zpos.png' ) }));
materialArray.push(new THREE.MeshBasicMaterial( { map: THREE.ImageUtils.loadTexture( 'images/dawnmountain-zneg.png' ) }));
var skyboxGeom = new THREE.CubeGeometry( 5000, 5000, 5000, 1, 1, 1, materialArray );
var skybox = new THREE.Mesh( skyboxGeom, new THREE.MeshFaceMaterial() );
skybox.flipSided = true;
scene.add( skybox );
```

Creating an environment map

- Take a panoramic photo
- Load it into Blender (e.g.) and map it onto a sphere
- Create 6 images with orthographic cameras
- Tutorial

How to do this?



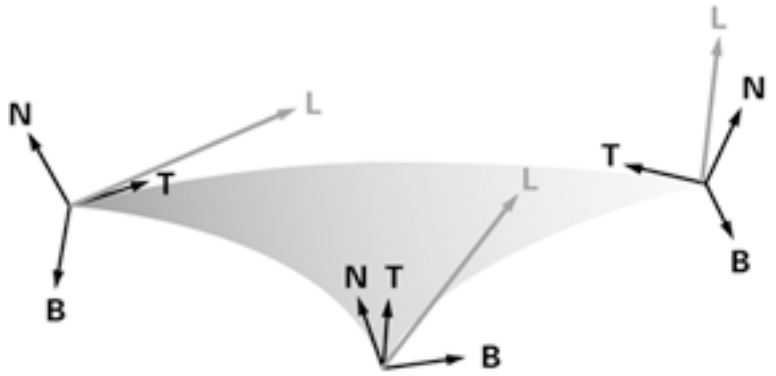
Bump Mapping / Normal Mapping

- Bump map (aka UV map) is an array of values for modulating surface normals
- XYZ components stored in RGB, 128=0

- Details
- How to generate a normal map from height field?

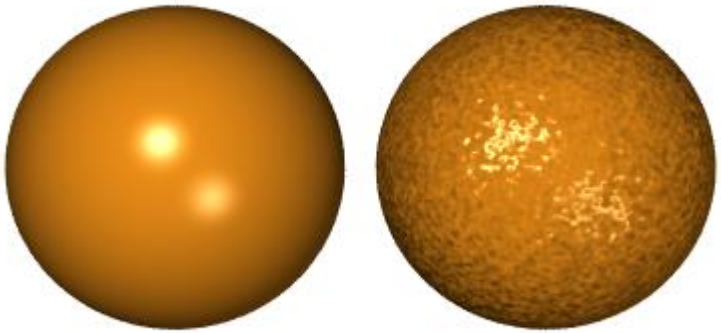
Bump Mapping Math

- Light vector, etc. must be in the same coordinate system (frame) as normals
- Could convert everything back to object space or (easier) rotate direction vectors into texture space
  - Called *texture-space bump mapping* (aka *tangent-space bump mapping*)
  - Requires a new transformation matrix for every point
  - Allows you to use a single bump map for multiple models



Bump mapping - mapowanie wypukłości

- Technika polega na zaburzeniu kierunku wektora normalnego i w konsekwencji zaburzeniu równomiernego oświetlenia i sprawieniu wrażenia wypukłości na gładkiej powierzchni.



Bump mapping – dwie techniki

- Techniki wiążą się z ustaleniem źródła zaburzeń normalnych
- Pierwsza technika odwołuje się do pracy Jamesa F. Blinna z 1978 r. : Simulation of Wrinkled Surfaces https://www.microsoft.com/en-us/research/wp-content/uploads/1978/01/p286-blinn.pdf . Oparta jest na teksturach.
- Metoda Blinna została ponownie przeanalizowana i rozszerzona w pracy Mikkelsena: Simulation of Wrinkled Surfaces Revisited (2008).
- Druga technika bezpośrednio zaburza normalne metodą proceduralną.

Bump mapping - technika Blinna

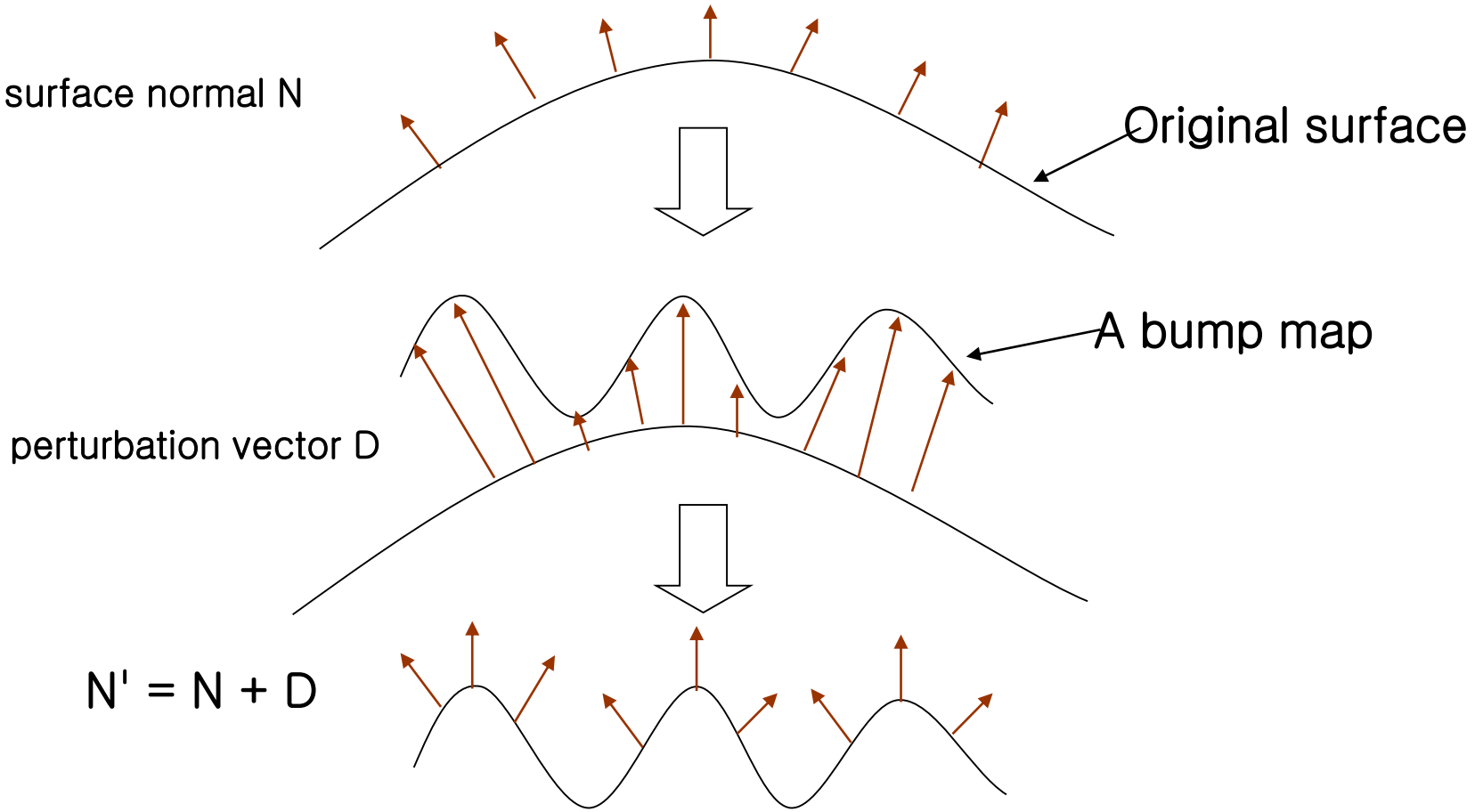
- Technika Blinna wykorzystuje monochromatyczną teksturę, potraktowaną jako mapę wysokości.

Bump Mapping – kroki obliczeń

- Odczytaj wysokość z mapy wysokości w punkcie odpowiadającym położeniu na powierzchni.
- Oblicz wektor normalny do mapy wysokości (zwykle używa się metody różnic skończonych).
- Połącz obie normalne – tę z mapy wysokości i z rzeczywistej powierzchni.
- Oblicz oświetlenie według nowych normalnych.

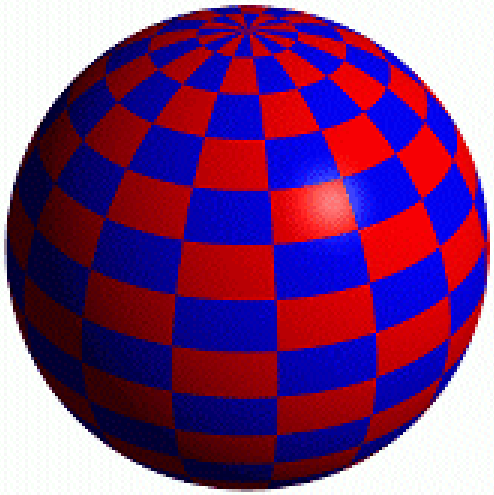
- Poglądowy schemat na następnym slajdzie.

Bump mapping - technika Blinna

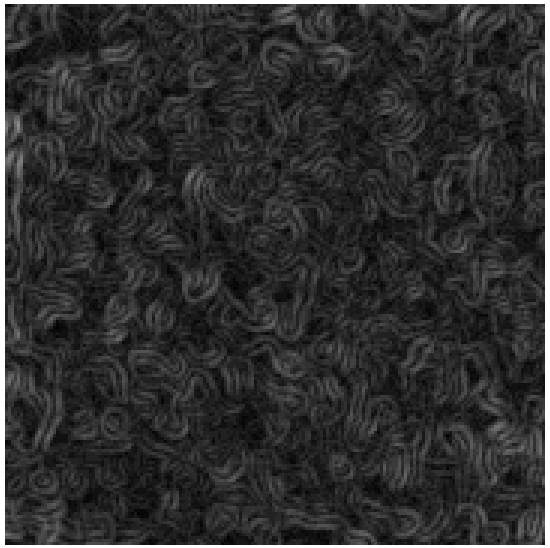


Przykład bump mappingu

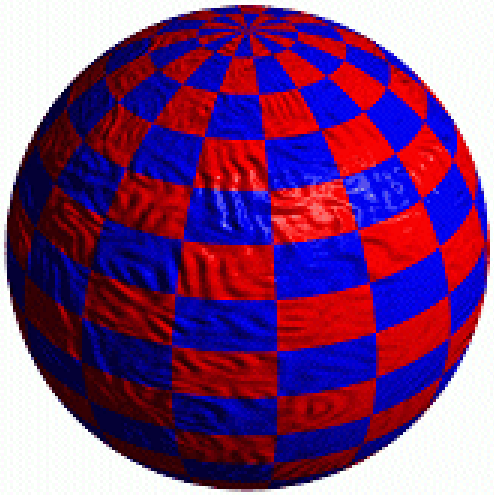
- W tym przykładzie mapa wysokości łączona jest z nakładaną teksturą, co jest typowym zabiegiem.



Sphere w/ diffuse texture



Swirly bump map



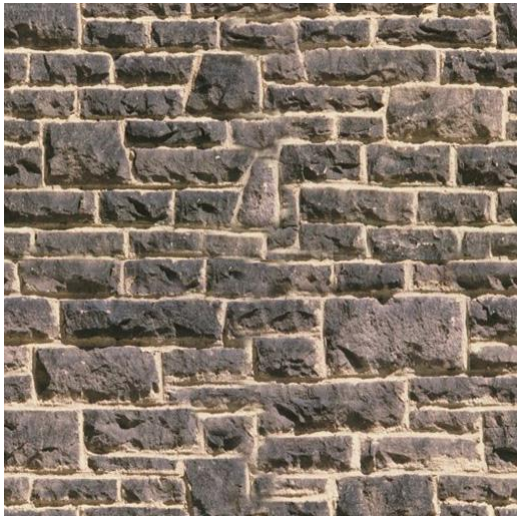
Sphere w/ diffuse texture and swirly bump map

Bump mapping w three.js

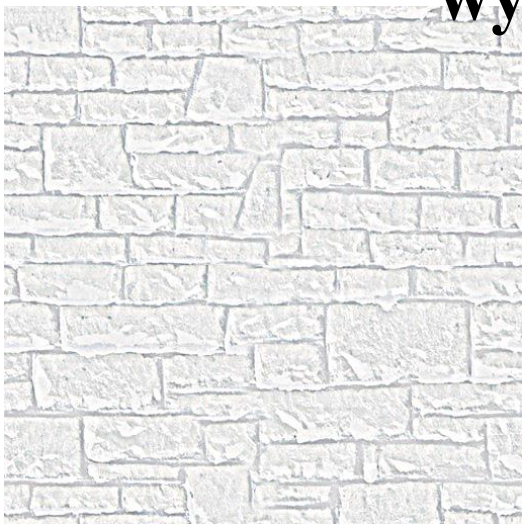
```
var texture =
THREE.ImageUtils.loadTexture("stone.jpg");

var mat = new THREE.MeshPhongMaterial();
mat.map = texture;
var bump =
THREE.ImageUtils.loadTexture("s_bump.jpg");
mat.bumpMap = bump;
mat.bumpScale = 0.2;
var mesh = new THREE.Mesh(geom, mat);
```

Bump mapping – oparty na mapie wysokości

- Często mapa wysokości jest tworzona jest na podstawie mapowanej tekstury.
- Przykład: http://www.smartjava.org/ltjs/chapter-10/02-bump-map.html



To jest mapa wysokości



Reflect/bump Vertex Shader

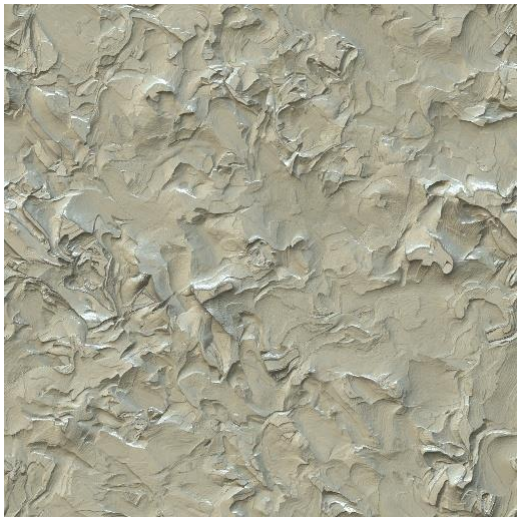
```
attribute vec3 g_Position, g_TexCoord0, g_Tangent, g_Binormal,
    g_Normal;
uniform mat4 world, worldInverseTranspose, worldViewProj,
    viewInverse;
varying vec2 texCoord;
varying vec3 worldEyeVec, worldNormal, worldTangent, worldBinorm;

void main() {
  gl_Position = worldViewProj * vec4(g_Position.xyz, 1.);
  texCoord.xy = g_TexCoord0.xy;
  worldNormal = (worldInverseTranspose * vec4(g_Normal, 1.)).xyz;
  worldTangent = (worldInverseTranspose *
       vec4(g_Tangent, 1.)).xyz;
  worldBinorm = (worldInverseTranspose *
       vec4(g_Binormal, 1.)).xyz;
  vec3 worldPos = (world * vec4(g_Position, 1.)).xyz;
  worldEyeVec = normalize(worldPos - viewInverse[3].xyz);
}
```

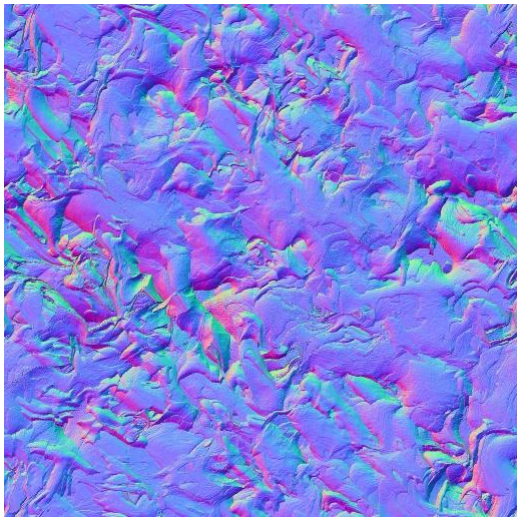
Reflect/bump Fragment Shader

```
const float bumpHeight = 0.2;
uniform sampler2D normalSampler;
uniform samplerCube envSampler;
varying vec2 texCoord;
varying vec3 worldEyeVec, worldNormal, worldTangent, worldBinorm;

void main() {
  vec2 bump = (texture2D(normalSampler, texCoord.xy).xy *
       2.0 - 1.0) * bumpHeight;
  vec3 normal = normalize(worldNormal);
  vec3 tangent = normalize(worldTangent);
  vec3 binormal = normalize(worldBinorm);
  vec3 nb = normal + bump.x * tangent + bump.y * binormal;
  nb = normalize(nb);
  vec3 worldEye = normalize(worldEyeVec);
  vec3 lookup = reflect(worldEye, nb);
  vec4 color = textureCube(envSampler, lookup);
  gl_FragColor = color; }
```

Normal mapping

- http://www.smartjava.org/ltjs/chapter-10/03-normal-map.html





Normal mapping w three.js

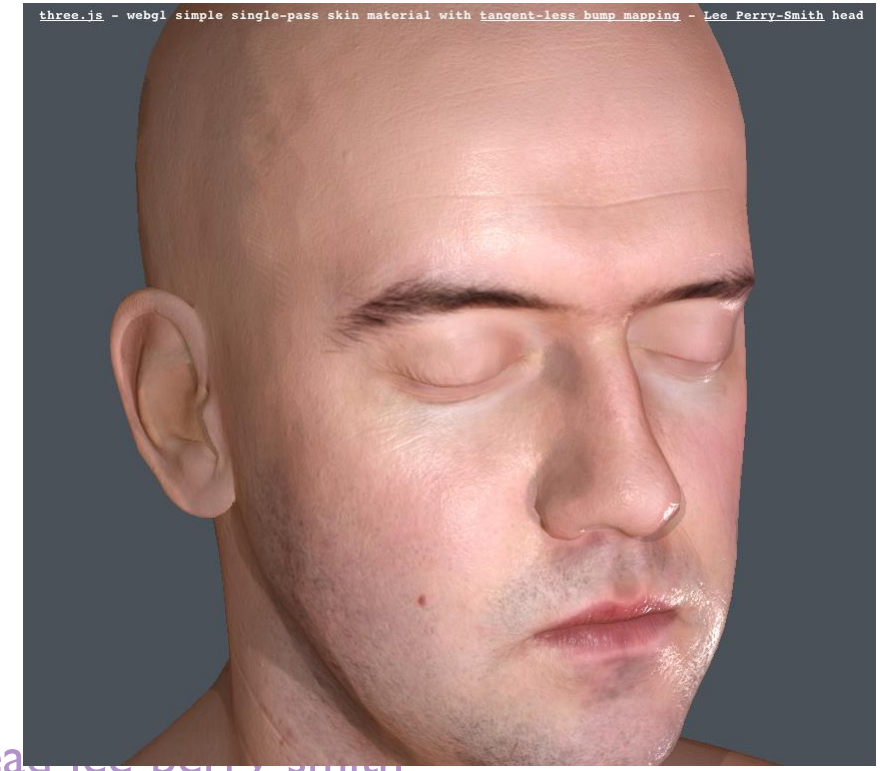
```
var tex =
THREE.ImageUtils.loadTexture("  ");

var mat =
THREE.ImageUtils.loadTexture(„normal.jpg");


var mat2 = new THREE.MeshPhongMaterial();
mat2.map = tex;
mat2.normalMap = mat;

var mesh = new THREE.Mesh(geom, mat2);
```

Jak zrobić mapę normalnych dla danej tekstury

- Wiele różnych podejść. Nie dla każdej tekstury udaje się znaleźć poprawną mapę normalnych.
- W programie Gimp trzeba doładować wtyczkę normalmap: http://registry.gimp.org/node/69

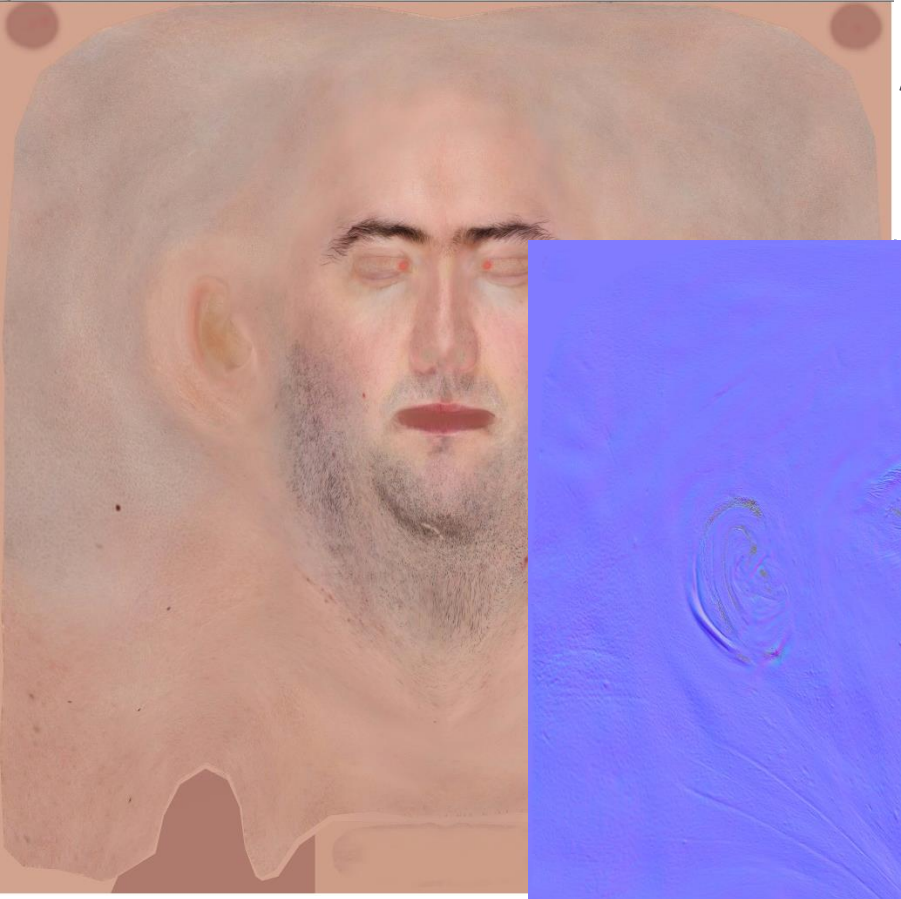
Przykład Bump Mappingu w Three.js

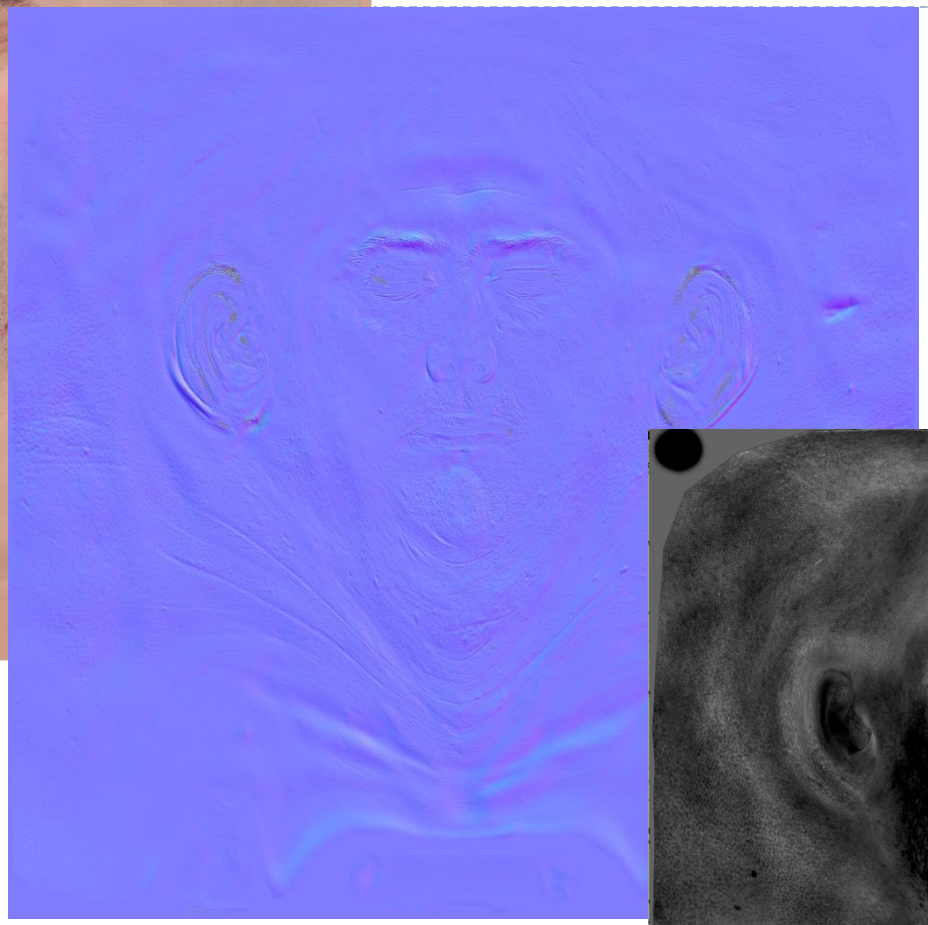
- 3D Head Scan by Lee Perry-Smith



- https://sketchfab.com/3d-models/head-lee-perry-smith-4d07eb2030db4406bc7eee971d1d3a97

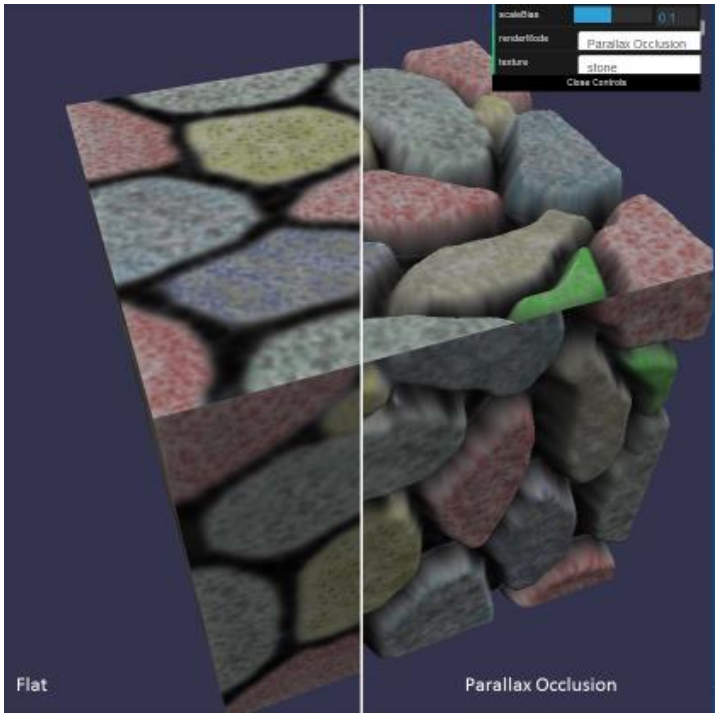
Tekstura, mapa normalnych i mapa wypukłości



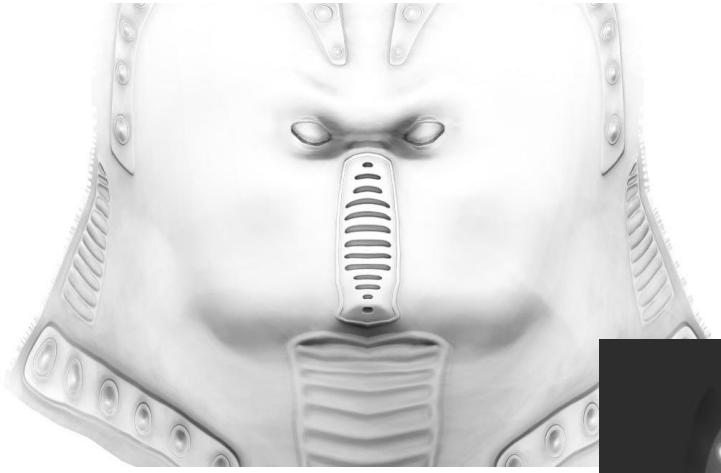


Parallax mapping

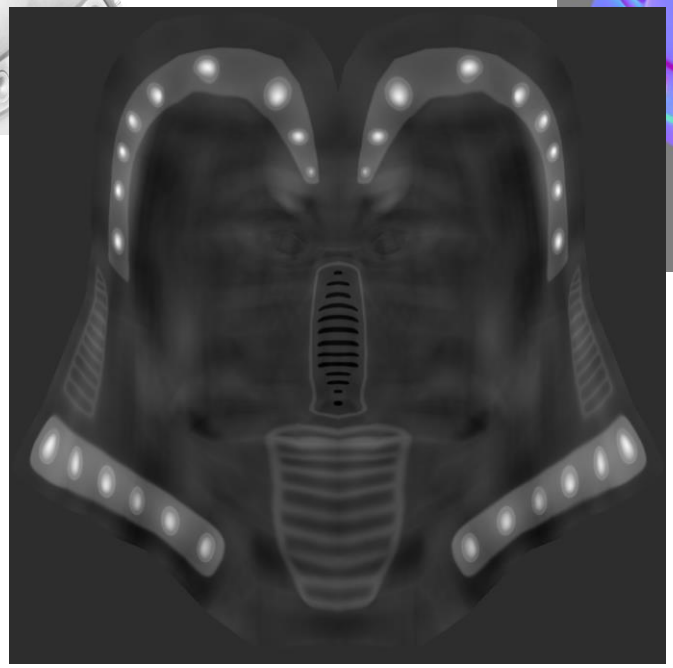
- Rozszerzenie Bump Mappingu / Normal Mappingu.
- Polega na dynamicznym modyfikowaniu tekstury w zależności od kąta patrzenia, np. na translacji jasnych tekseli
- Porównanie:



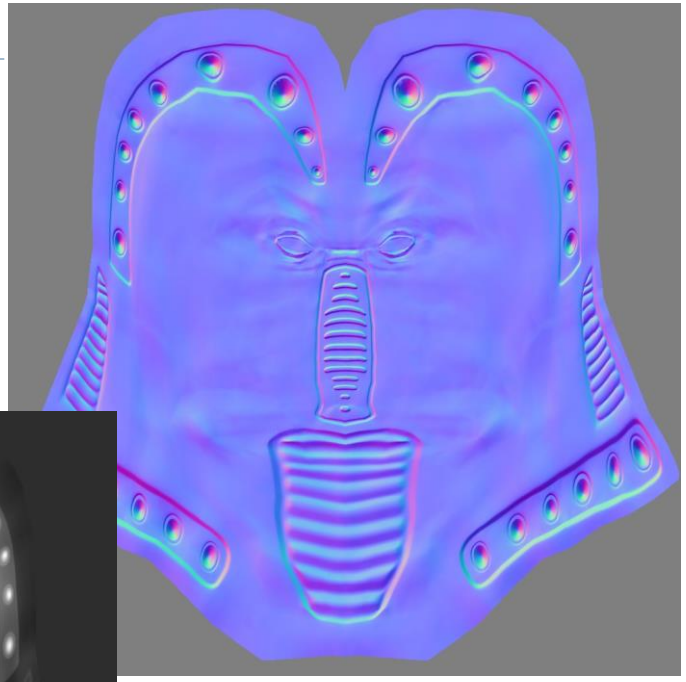
Displacement Mapping



Texture map



Displacement map



Normal map

Alter location of vertices in vertex shader

How to do this?

How exactly are the two colors combined?

What if underlying pixel is also transparent?

Blending – low level WebGl

- Blend a *source* and *destination* fragment
  - *Source:* fragment currently being drawn
  - *Destination:* fragment already in frame buffer
- Turn on blending
  - `gl.enable(gl.BLEND);`
  - `gl.disable(gl.DEPTH_TEST); //optional`
  - `gl.blendFunc(gl.SRC_ALPHA, gl.ONE);`
    - First parameter: *source factor*
    - Second parameter: *destination factor*
    - Possible values: ZERO, ONE, SRC_COLOR, ONE_MINUS_SOURCE_COLOR, SRC_ALPHA, ONE_MINUS_SRC_ALPHA, DST_ALPHA, ONE_MINUS_DST_ALPHA

Blending Function

- Source: R_s, G_s, B_s, A_s
- Destination: R_d, G_d, B_d, A_d
- Source factor: S_r, S_g, S_b, S_a (Can have separate RGBA components of source, destination factors)
- Dest factor: D_r, D_g, D_b, D_a
- Combining colors:
  - $R_{result} = R_s * S_r + R_d * D_r$
  - $G_{result} = G_s * S_g + G_d * D_g$
  - $B_{result} = B_s * S_b + B_d * D_b$
  - $A_{result} = A_s * S_a + A_d * D_a$

Blending example

- Draw a transparent polygon (given by alpha) on top of an opaque background
  - gl.blendFunc(gl.SRC_ALPHA, gl.ONE)
  - Resulting function:
    - $R_{result} = R_s * A_s + R_d$
- Source and background both potentially transparent
  - gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA)
    - $R_{result} = R_s * A_s + R_d * (1-A_s)$
    - Does this really work?

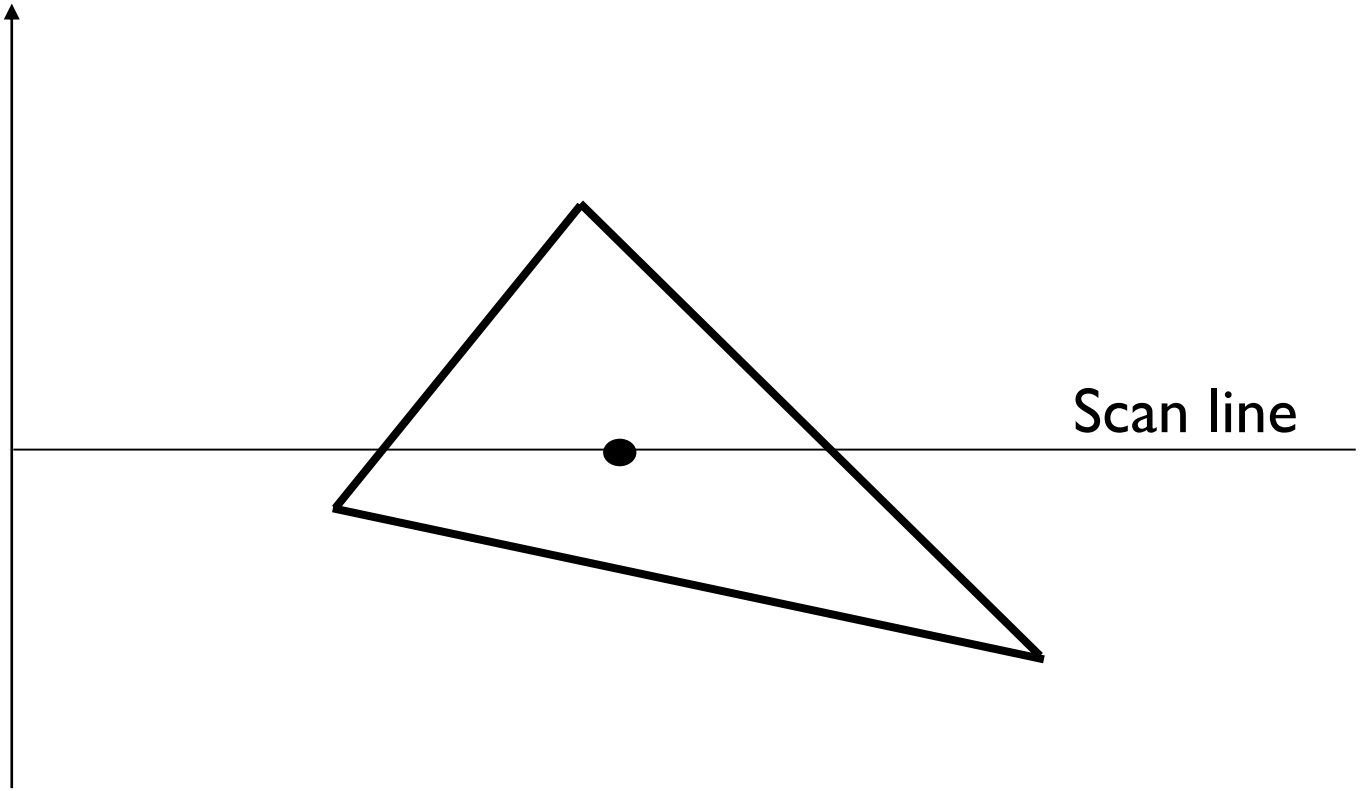
Blending Problems

- What if we leave the depth test on and want to draw these faces (back to front order):
  1. (1, 0, 0, .5) [Red 50% opaque]
  2. (0, 1, 0, 1) [Green 100% opaque]
  3. (0, 0, 1, .5) [Blue 50% opaque]
- What happens if we draw faces…
  - 1, then 2, then 3
  - 2, then 1, then 3
  - 1, then 3, then 2

Blending Problems

- Unfortunately, blending doesn't work with depth-buffering and arbitrary polygon order
- What to do?
- Possible solution:
  - Draw all opaque polygons with depth-buffering
  - Enable blending, sort transparent polygons, draw back to front
  - Or skip the sorting

1st idea: Gouraud interp. of texcoords



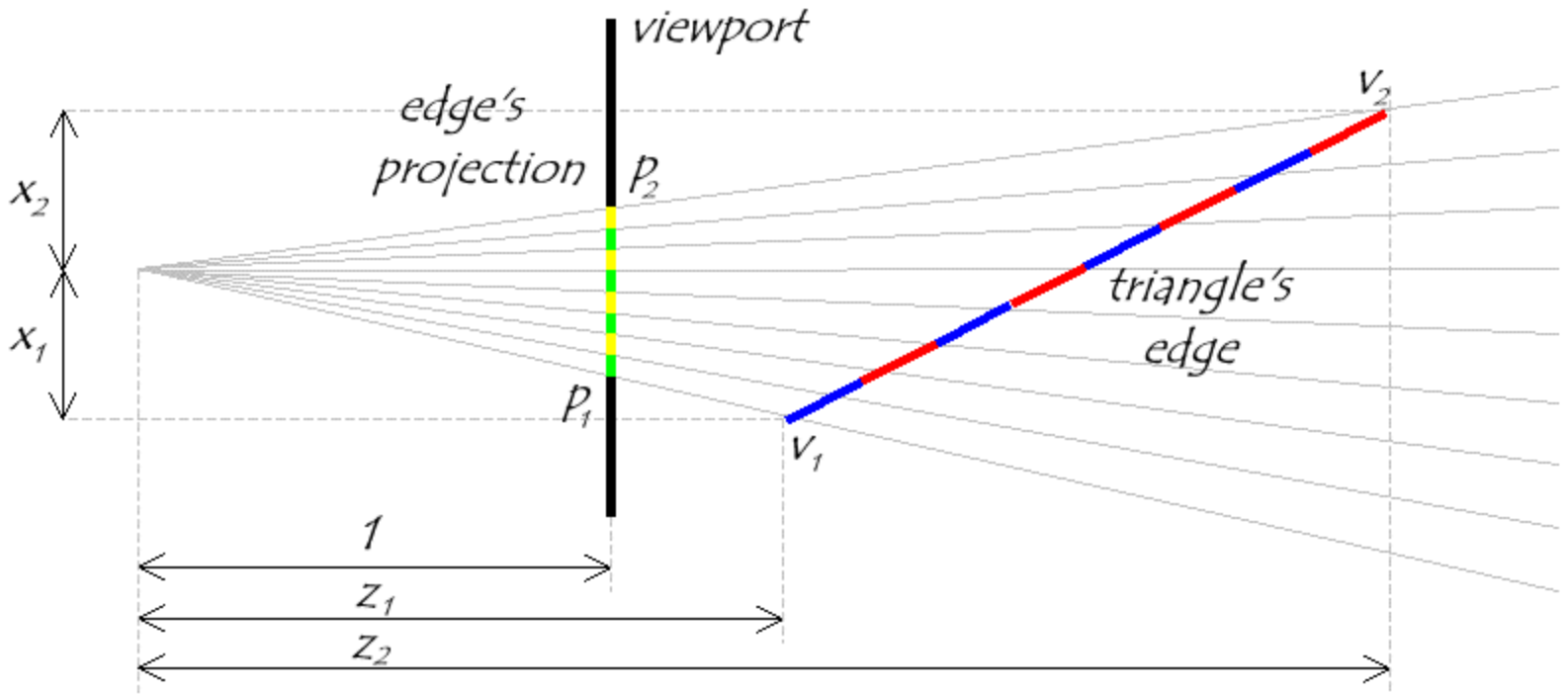
Actual implementation efficient: difference equations while scan converting

Artifacts

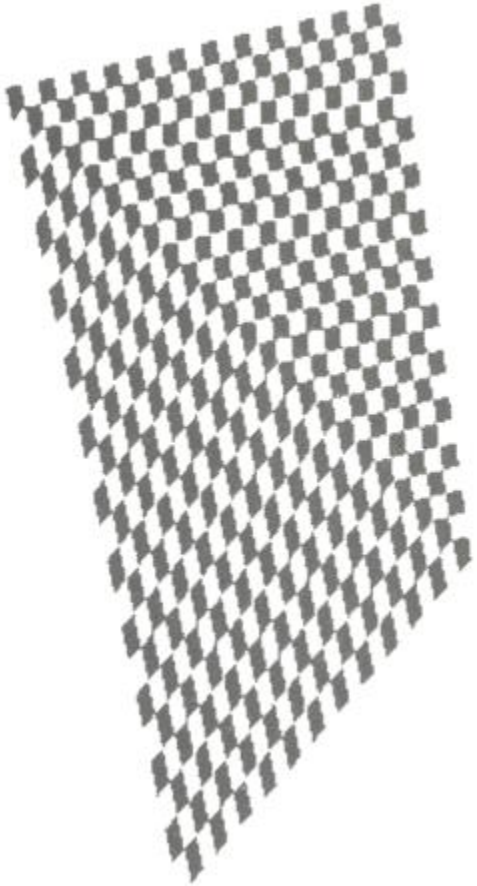
- McMillan's demo of this is at
  http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide05.html
- Another example
  http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide06.html
- What artifacts do you see?
- Why?
- Why not in standard Gouraud shading?
- Hint: problem is in interpolating parameters
- Wikipedia page

Interpolating Parameters

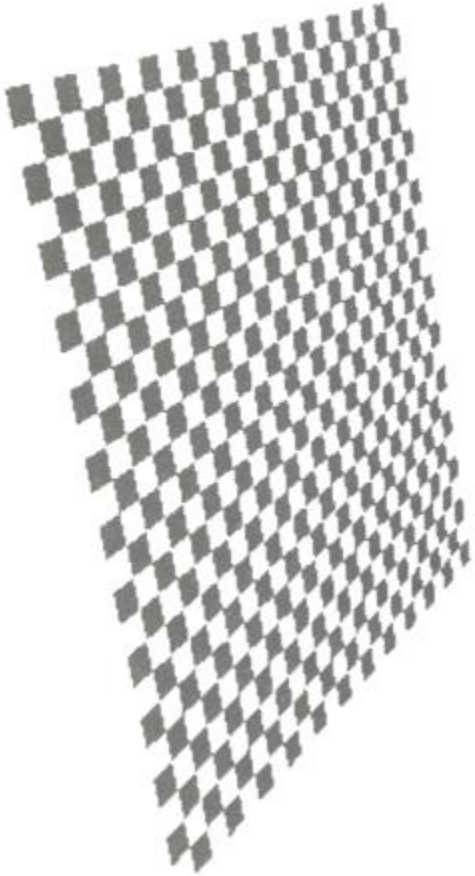
- The problem turns out to be fundamental to interpolating parameters in screen-space
  - *Uniform steps in screen space ≠ uniform steps in world space*



Texture Mapping



Linear interpolation
of texture coordinates



Correct interpolation
with perspective divide

Hill Figure 8.42

Interpolating Parameters

- Perspective foreshortening is not getting applied to our interpolated parameters
  - Parameters should be compressed with distance
  - Linearly interpolating them in screen-space doesn't do this

Perspective-Correct Interpolation

- Skipping a bit of math to make a long story short…
  - Rather than interpolating *u* and *v* directly, interpolate *u*/*z* and *v*/*z*
    - These do interpolate correctly in screen space
    - Also need to interpolate *z* and multiply per-pixel
  - Problem: we don't know *z* anymore
  - Solution: we do know $w \sim 1/z$
  - So…interpolate *uw* and *vw* and *w*, and compute $u = uw/w$ and $v = vw/w$ for each pixel
    - This unfortunately involves a divide per pixel
- http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide14.html

Texture Map Filtering

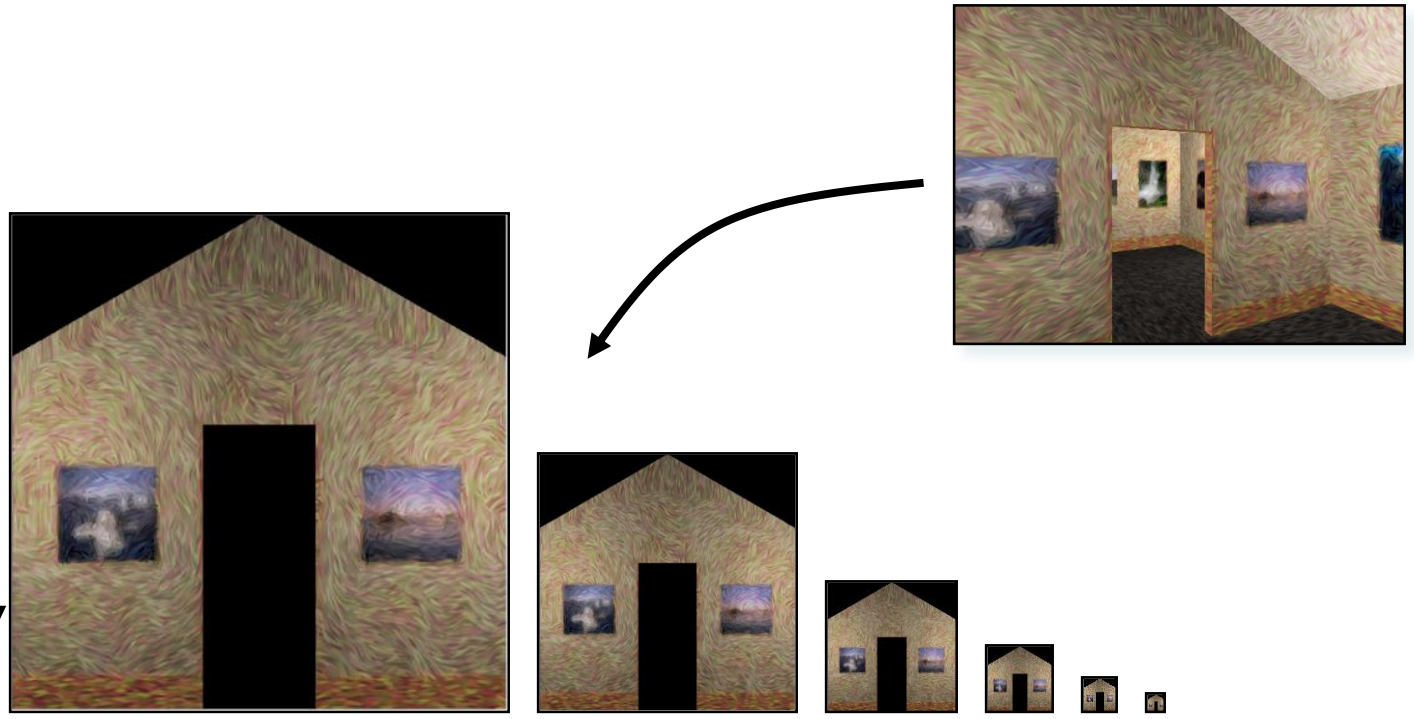
- Naive texture mapping aliases badly
- Look familiar?

```
int uval = (int) (u * denom + 0.5f);
int vval = (int) (v * denom + 0.5f);
int pix = texture.getPixel(uval, vval);
```

- Actually, each pixel maps to a region in texture
  - |PIX| < |TEX|
    - Easy: interpolate (bilinear) between texel values
  - |PIX| > |TEX|
    - Hard: average the contribution from multiple texels
  - |PIX| ~ |TEX|
    - Still need interpolation!

Mip Maps

- Keep textures prefiltered at multiple resolutions
  - For each pixel, linearly interpolate between two closest levels (e.g., trilinear filtering)
  - Fast, easy for hardware
- Why



MIP-map Example

- No filtering:

AAAAAAAGH
MY EYES ARE BURNING

- MIP-map texturing:

Where are my glasses?

Outline

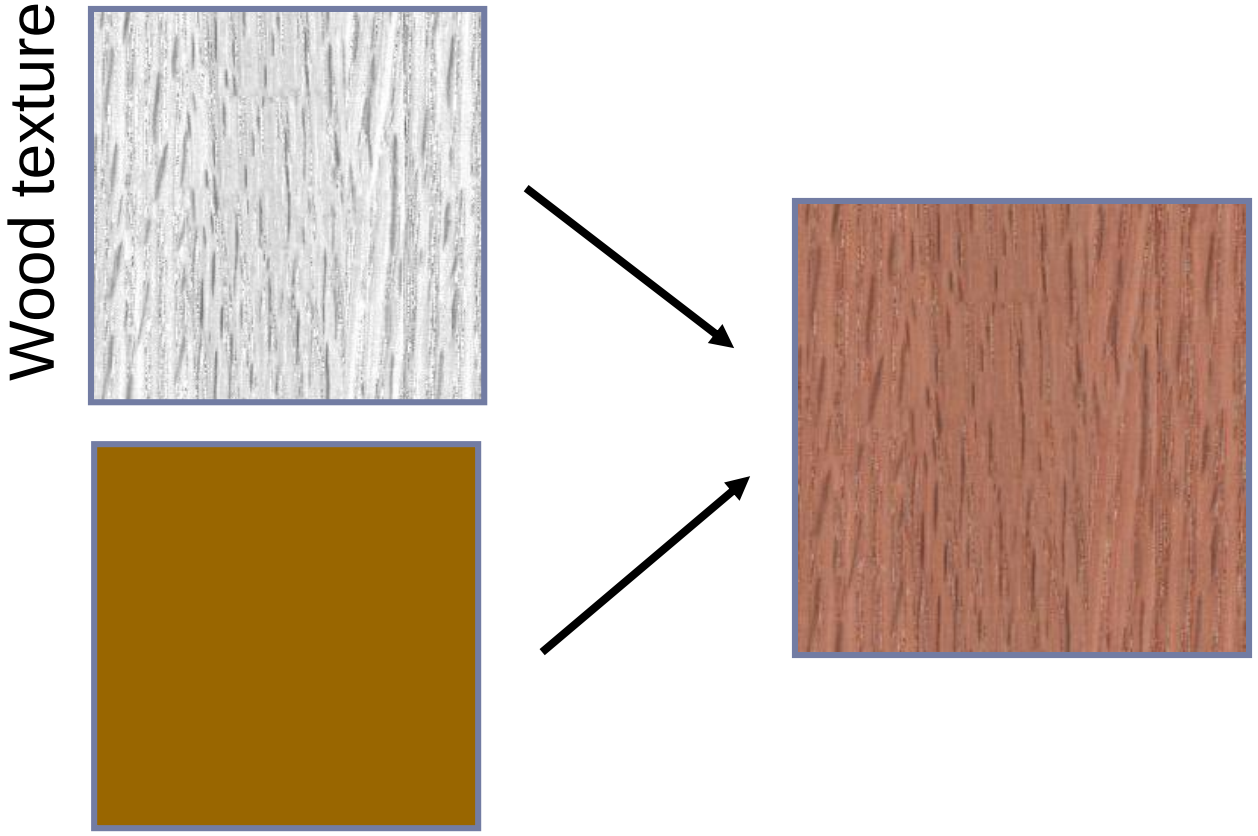
- Types of projections
- Interpolating texture coordinates
- *Broader use of textures*

Texture Mapping Applications

- Modulation, light maps
- Bump mapping
- Displacement mapping
- Illumination or Environment Mapping
- Procedural texturing
- And many more

Modulation textures

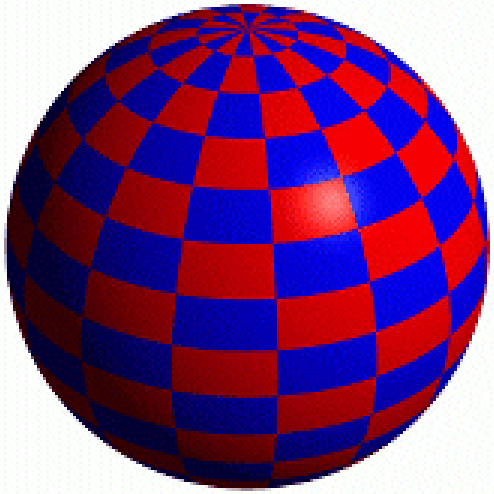
Map texture values to scale factor



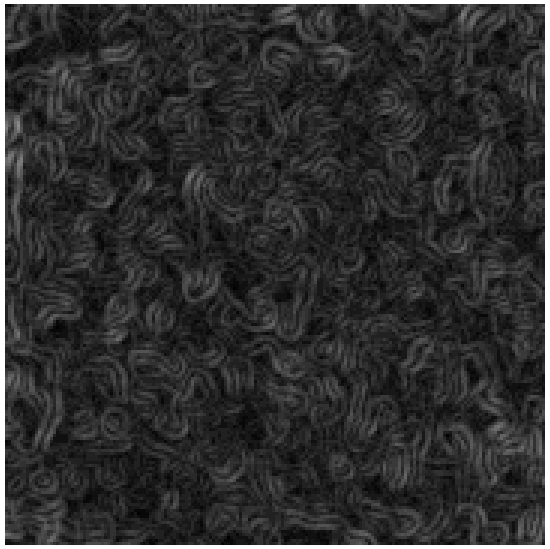
$$I = T(s,t)(I_E + K_A I_A + \sum_L (K_D (N \bullet L) + K_S (V \bullet R)^n) S_L I_L + K_T I_T + K_S I_S)$$

Bump Mapping

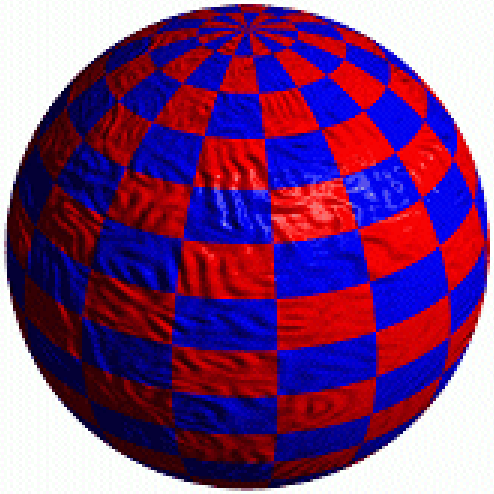
- Texture = change in surface normal!



Sphere w/ diffuse texture

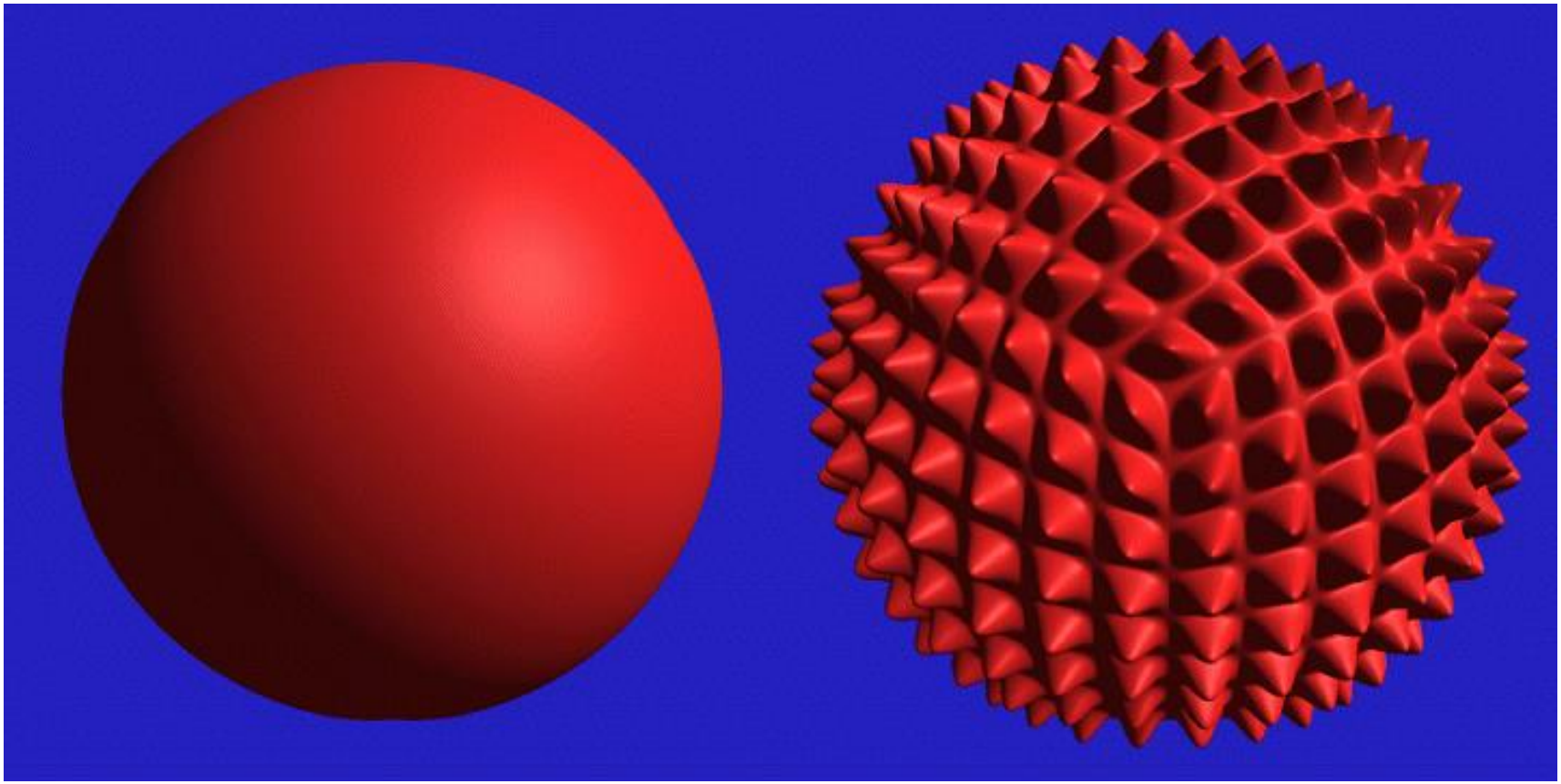


Swirly bump map



Sphere w/ diffuse texture and swirly bump map

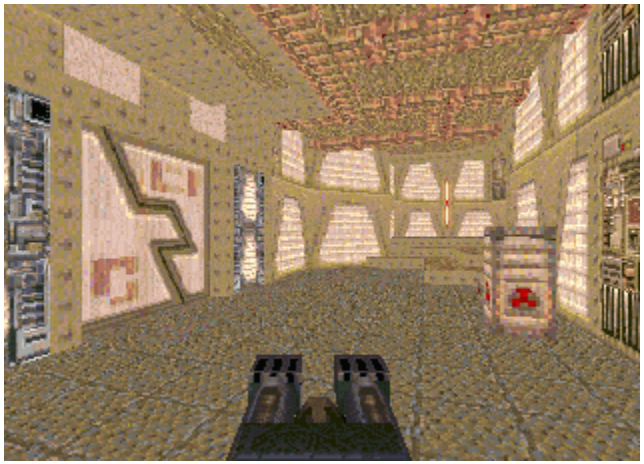
Displacement Mapping

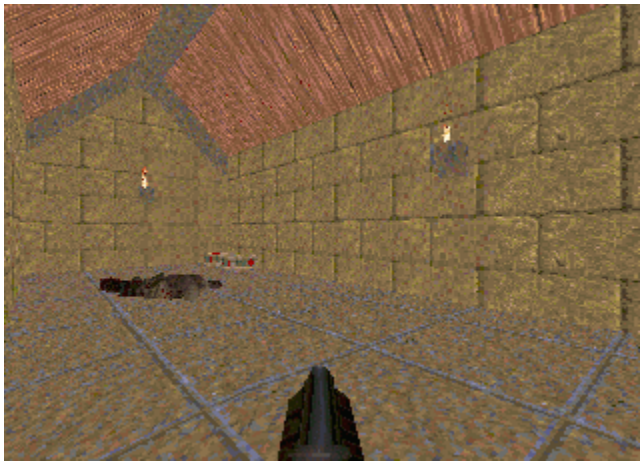


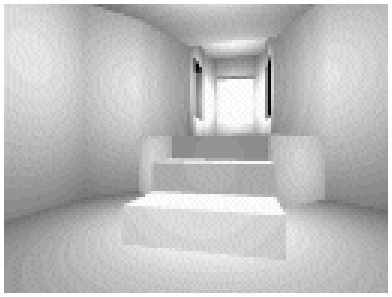
Illumination Maps

- Quake introduced *illumination maps* or *light maps* to capture lighting effects in video games

Texture map:







Light map

Texture map + light map:

Environment Maps

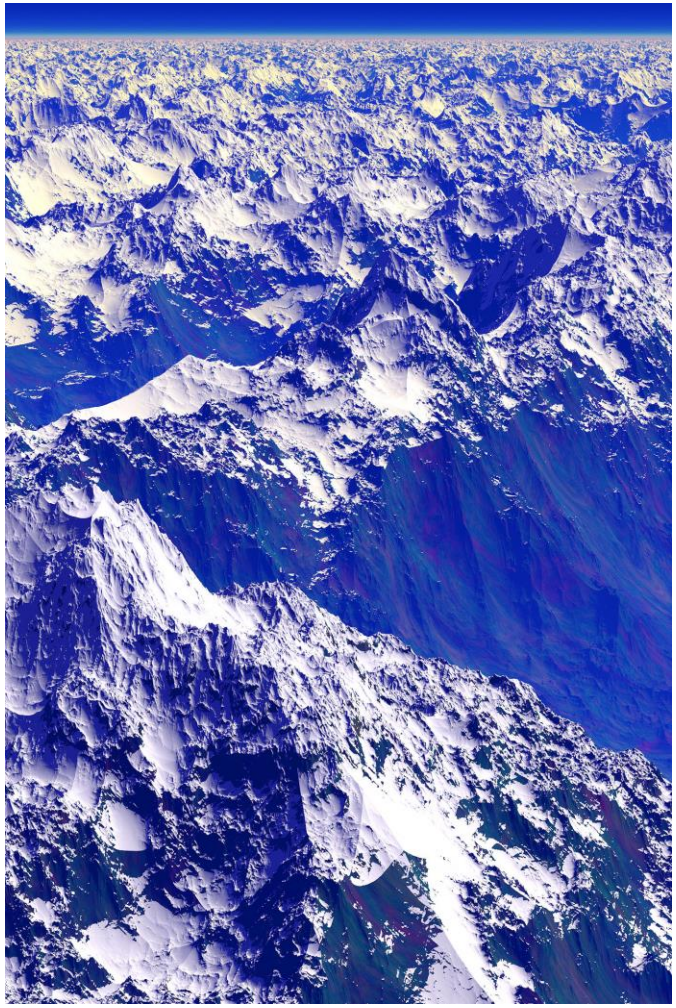
Images from *Illumination and Reflection Maps:*
Simulated Objects in Simulated and Real Environments
Gene Miller and C. Robert Hoffman
SIGGRAPH 1984 "Advanced Computer Graphics Animation" Course Notes

Solid textures

Texture values indexed by 3D location (x,y,z)

- Expensive storage, or
- Compute on the fly, e.g. Perlin noise →

Procedural Texture Gallery



Metallichron - Brassy Rise

Texture mapping

by Armands Auseklis