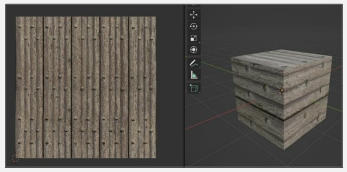
Shrinkwrapping

- directly map the surface to the image texture
- suitable for simple shapes, such as
  - cube
  - sphere
  - cylinder
  - infinite cylinder

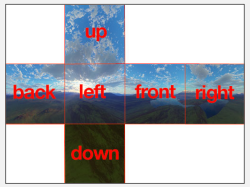
https://www.bigshrink.com/bulk-shrink-wrap-32-100-7

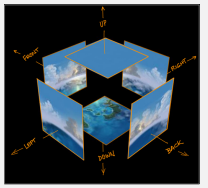
Shrinkwrapping Cubes

- apply image to each face using plane projection



- use a cubemap

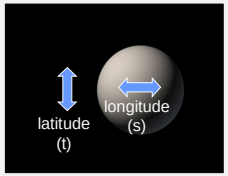




https://i.all3dp.com/workers/images/fit=scale-down,w=1200,h=630,gravity=0.5x0.5,format=jpeg/wp-content/uploads/2023/03/20181630/cube-mesh-with-medieval-wooden-texture-aftab-ali-via-all3dp-230123.jpg
https://en.wikipedia.org/wiki/Cube_mapping
https://scalibq.wordpress.com/2013/06/23/cubemaps/

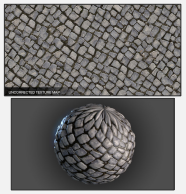
Shrinkwrapping Spheres

$$x = r \cos(lat) \sin(long)$$
$$y = r \sin(lat)$$
$$z = r \cos(lat) \cos(long)$$



latitude is between –90 and 90
longitude is between –180 and 180

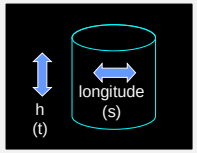
- convert (x,y,z) to (long,lat)
- map long → s, lat → t



https://i.sstatic.net/oX2gX.jpg

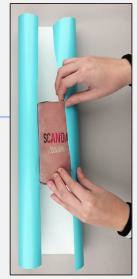
Shrinkwrapping Cylinders

$$x = r \sin(long)$$
$$y = h$$
$$z = r \cos(long)$$



h is between –height/2 and height/2
longitude is between –180 and 180

- convert (x,y,z) to (long,h)
- map long → s, h → t

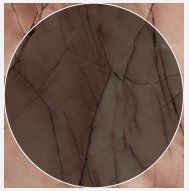


https://www.wikihow.com/Wrap-Cylindrical-Gifts#/Image:Wrap-Cylindrical-Gifts-Step-6-Version-3.jpg

Cylinder Caps

idea #1: compute (long,h,r) for cap points, then ignore r

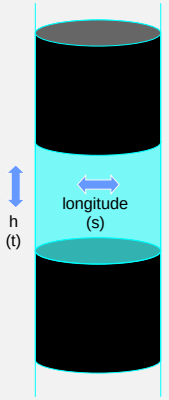
idea #2: lay texture over cap and cut out the circle



- map x → s, z → t
  - x, z are in the range -radius to radius

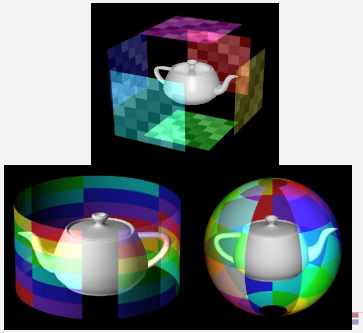
Shrinkwrapping Infinite Cylinders

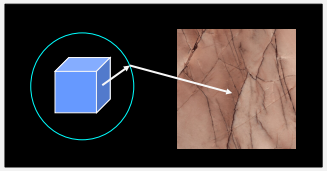
- compute longitude as in shrinkwrap approach
- for h, use mod to bring into range – height/2 to height/2



Intermediate Surfaces (Map Shapes)

- map OC point on a complex object to a simpler map shape (cube, sphere, cylinder), then compute texture coordinates for the map shape





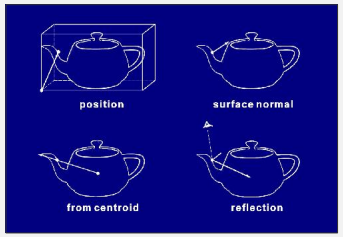
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Object → Map Shape

many strategies –

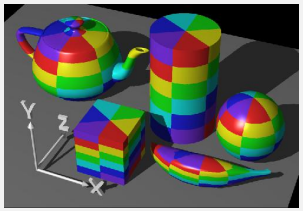
- convert point to shape coordinates
- ray-based – intersect ray with map shape
  - ray through point from corner of bounding box
  - surface normal at the point
  - ray through point from center of object
  - perfect reflection ray from viewer to point

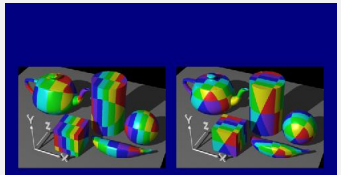
plug (x,y,z) surface point into equations for map shape



Examples

map shape: cylinder
method: convert point to shape coordinates (compute (long,h) for points)

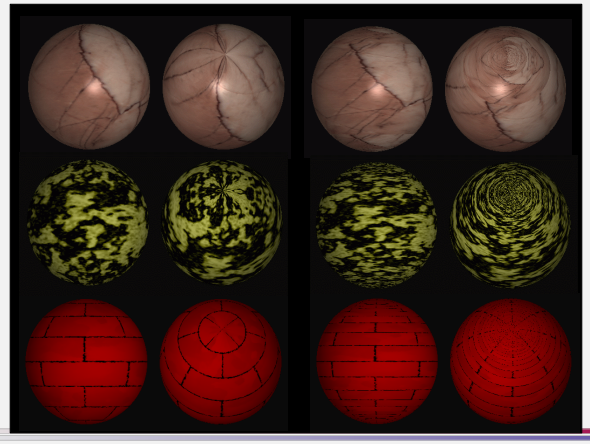




how is the map shape used in these pictures?

Examples

some combinations of map shape and object→map shape mapping work better than others



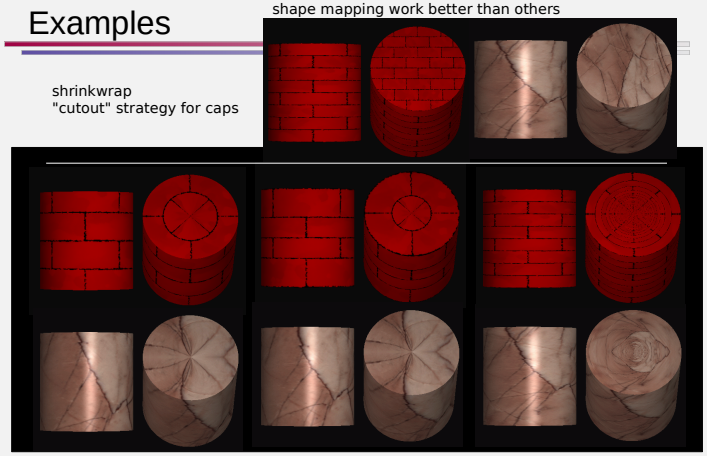
shrinkwrap

infinite cylinder map shape
ray through center

Examples

some combinations of map shape and object→map shape mapping work better than others

shrinkwrap
"cutout" strategy for caps



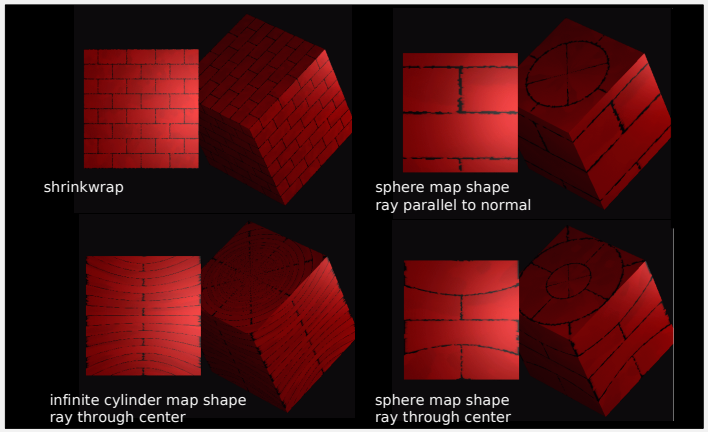
sphere map shape
ray parallel to normal

sphere map shape
ray through center

infinite cylinder map shape
ray through center

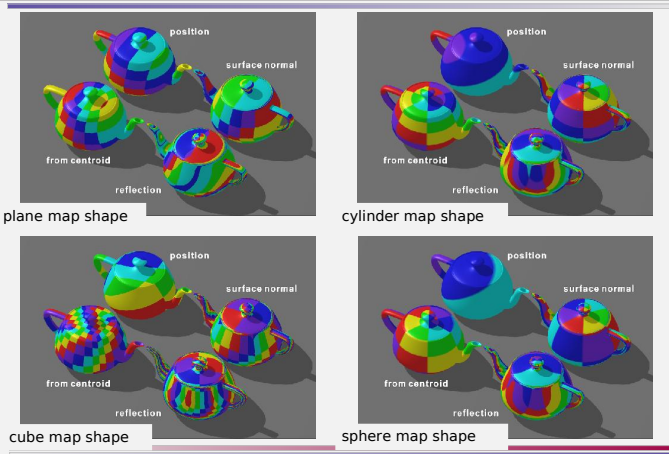
Examples

some combinations of map shape and object→map shape mapping work better than others



Examples

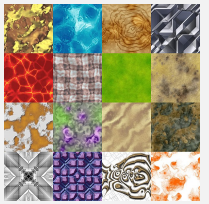
some combinations of map shape and object→map shape mapping work better than others



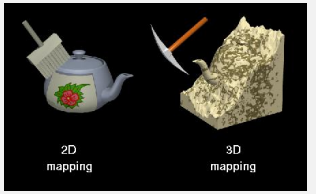
 http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Procedural Textures

- instead of sampling an image to obtain a texture color, apply an equation
  - computation is done in the fragment shader (per pixel)



- procedural textures can be 2D or 3D
- space-efficient but more time-consuming to compute



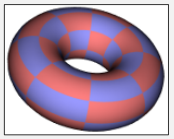
 https://commons.wikimedia.org/wiki/File:Tiling_procedural_textures.jpg http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/

Examples

GLSL `mod(x,y) = x-y*floor(x/y)`

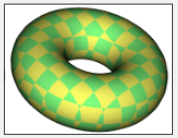
- 2D checkerboard
  - uses texture coordinates

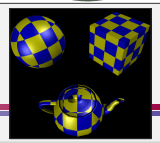
```
vec4 color;
float a = floor(v_texCoords.x * scale);
float b = floor(v_texCoords.y * scale);
if (mod(a+b, 2.0) > 0.5) {  // a+b is odd
    color = vec3(1.0, 0.5, 0.5, 1.0); // pink
}
else {  // a+b is even
    color = vec3(0.6, 0.6, 1.0, 1.0); // light blue
}
```



- 3D (space-filling) checkerboard
  - similar to 2D but including x, y, and z coordinates (OC)

```
float a = floor(v_objCoords.x * scale);
float b = floor(v_objCoords.y * scale);
float c = floor(v_objCoords.z * scale);
if (mod(a+b+c, 2.0) > 0.5) {  // a+b+c is odd
    color = vec3(0.5, 1.0, 0.5);
}
else {  // a+b+c is even
    color = vec3(1.0, 1.0, 0.4);
}
```



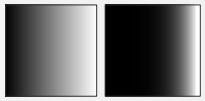


Creating Procedural Textures

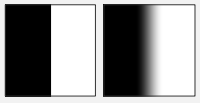
- common ingredients
  - gradients
  - overlaid patterns
  - noise

Gradients

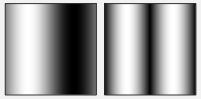
```
value = texcoords.x;                          // linear gradient
value = pow(texcoords.x,5.0);                 // power
value = step(0.5,texcoords.x);                // step
value = smoothstep(0.35,0.65,texcoords.x);    // smooth step
value = 0.5+sin(texcoords.x*3.1415927*2.0)*0.5;  // sin
value = abs(sin(texcoords.x*3.1415927*2.0));  // abs(sin)
```



- linear
- power



- step
- smooth step



- sin
- abs(sin)

```
float n = 8.0;
vec3 texcoords = mod(v_objcoords.xyz,1.0/n)*n;
```

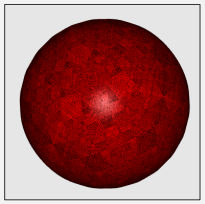
- repetition (stripes)

`vec4 texcolor = vec4(vec3(value),1.0);`

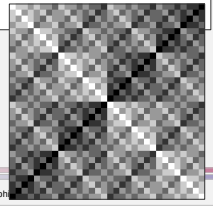
Overlaid Patterns

`vec4 texcolor = vec4(vec3(value),1.0);`

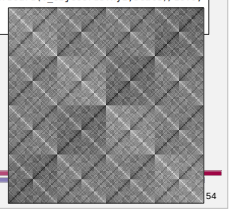
```
float checkerboard ( vec3 objcoords, float n ) {
  vec3 texcoords = step(0.5,mod(objcoords,1.0/n)*n);
  float value = step(0.5,mod(texcoords.x+texcoords.y+texcoords.z,2.0));
  return value;
}
```



```
float value = 0.0, scale = 1.0;
for ( int step = 0 ; step < 5 ; step++ ) {
    value += checkerboard(v_objcoords.xyz,scale)/5.0;
    scale *= 2.0;
}
```



```
float value = 0.0, scale = 1.0;
for ( int step = 0 ; step < 16 ; step++ ) {
    value += checkerboard(v_objcoords.xyz,scale)/16.0;
    scale *= 2.0;
}
```

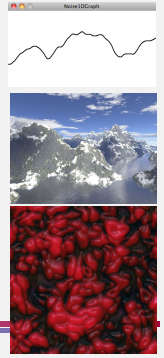


Natural Effects

- procedural textures are often used for materials such as wood, marble, granite, metal, stone
- commonly based on a regular function plus noise to create a random effect
  - e.g. Perlin noise, simplex noise

developed by Ken Perlin in 1983
published in SIGGRAPH in 1985
won Academy Award for Technical Achievement in 1997

key features
- pseudo-random (reproducible)
- coherent – values change smoothly
- rate of change can be controlled



https://natureofcode.com/book/introduction/
https://en.wikipedia.org/wiki/Perlin_noise
https://mrl.nyu.edu/~perlin/

Using Noise

- noise functions are not built in to GLSL – need a library
  - https://github.com/stegu/webgl-noise/ library used in the book's examples
  - https://github.com/stegu/psrdnoise/ newer version of some of the functions
- noise used directly (3D texture)

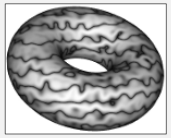
```
float value = snoise( scale*v_objCoords );
value = 0.75 + value*0.25;
color = vec3(1.0,value,1.0);
```

`snoise` produces a value between -1.0 and 1.0

- noise combined with a regular pattern (3D texture)

```
vec3 v = v_objCoords*scale;
float t = (v.x + 2.0*v.y + 3.0*v.z);
t += 1.5*snoise(v);
float value =  abs(sin(t));
color = vec3(sqrt(value));
```

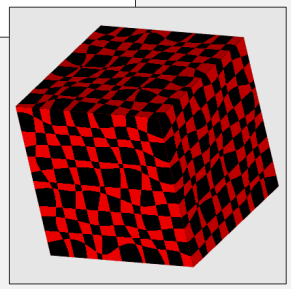
`abs(sin(t))`



Using Noise

```
float checkerboard ( vec3 objcoords, float n ) {
  vec3 texcoords = step(0.5,mod(objcoords,1.0/n)*n+0.4*snoise(objcoords));
  float value = step(0.5,mod(texcoords.x+texcoords.y+texcoords.z,2.0));
  return value;
}
```

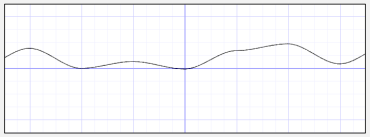
```
value = checkerboard(v_objcoords.xyz,3.0);

vec4 texcolor = vec4(vec3(value),1.0);
```



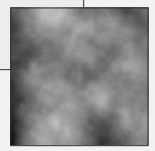
Fractal Brownian Motion

- add different iterations of noise, incrementing frequencies and decreasing amplitude each time

```
#define OCTAVES 6
float fbm (in vec2 st) {
    // Initial values
    float value = 0.0;
    float amplitude = .5;
    float frequency = 0.;
    //
    // Loop of octaves
    for (int i = 0; i < OCTAVES; i++) {
        value += amplitude * noise(st);
        st *= 2.;
        amplitude *= .5;
    }
    return value;
}
```



```
texcolor =
fbm(texcoords.xy*3.0);
```

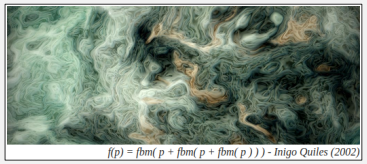


from *The Book of Shaders*, Patricio Gonzalez Vivo & Jen Lowe

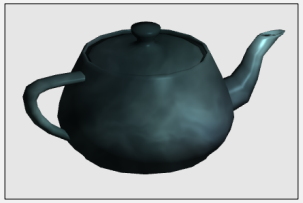
Fractal Brownian Motion

- can apply an fbm noise function to warp the texture coordinates themselves

```
vec2 q = vec2(0.);
q.x = fbm( st + 0.00*u_time);
q.y = fbm( st + vec2(1.0));

vec2 r = vec2(0.);
r.x = fbm( st + 1.0*q + vec2(1.7,9.2)+ 0.15*u_time );
r.y = fbm( st + 1.0*q + vec2(8.3,2.8)+ 0.126*u_time);

float f = fbm(st+r);

color = mix(vec3(0.101961,0.619608,0.666667),
            vec3(0.666667,0.666667,0.498039),
            clamp((f*f)*4.0,0.0,1.0));

color = mix(color,
            vec3(0,0,0.164706),
            clamp(length(q),0.0,1.0));

color = mix(color,
            vec3(0.666667,1,1),
            clamp(length(r.x),0.0,1.0));

gl_FragColor = vec4((f*f*f+.6*f*f+.5*f)*color,1.);
```



f(p) = fbm(p + fbm(p + fbm(p))) - Inigo Quiles (2002)



from *The Book of Shaders*,
Patricio Gonzalez Vivo & Jen Lowe