CS-184: Computer Graphics

Lecture 16: Texture and Other Maps

Maneesh Agrawala
University of California, Berkeley

Slides based on those of James O'Brien, Steve Marschner and Greg Humphreys

Announcements

Assignment 5: due Fri Nov 5 by 11pm

Today

Assigning Texture Coordinates
Distortion
Antialiasing
Bump & Displacement Maps, Environment Maps, Shadow Maps
Procedural Textures

4

Texture Mapping

Objects have properties that vary across the surface

5

Texture Mapping

So we make the shading parameters vary across the surface

[Foley et al. / Perlin]

6

Texture Mapping

Adds visual complexity; makes appealing images

[Pixar / Toy Story]

7

Surface Detail

Representing all detail in image with polygons would be cumbersome

8

Definition

Texture mapping: a technique of defining surface properties (especially shading parameters) in such a way that they vary as a function of position on the surface.

9

Examples

Wood gym floor with smooth finish

- diffuse color k_D varies with position
- specular properties k_S, n are constant

Glazed pot with finger prints

- diffuse and specular colors k_D, k_S are constant
- specular exponent n varies with position

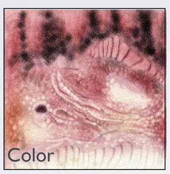
Adding dirt to painted surfaces

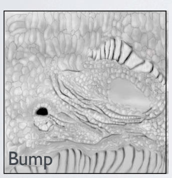
Simulating stone, fabric, …

- to approximate effects of small-scale geometry

More than Diffuse Color

Use a 2D image and map it to the surface of an object





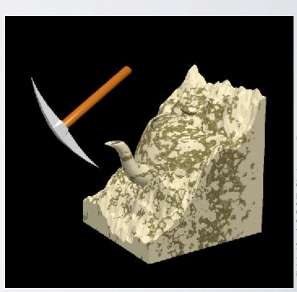

Mapping textures to surfaces

Usually the texture is an image (function of ***u***, v)

- Big question: where on the surface does the image go?
- Obvious only for a flat rectangle the same shape as the image
- Otherwise more interesting

Note that ***3D textures*** also exist

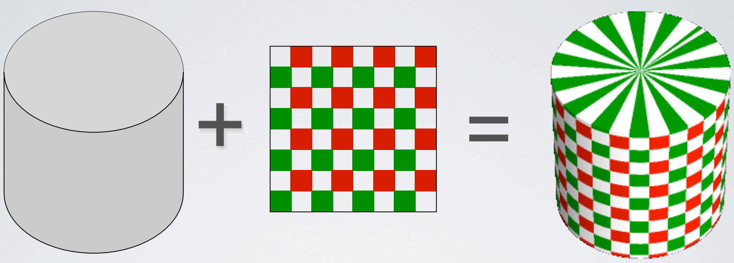
- Texture is a function of (*u*, *v*, *w*)
- Can just evaluate texture at 3D surface point
- Good for solid materials
- Often defined procedurally



[Wolfe / SG97 Slide set]

Assigning Texture Coordinates

Parameterization



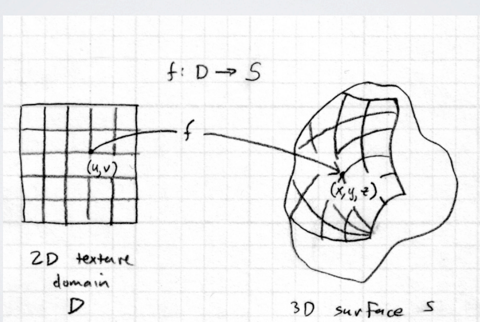
geometry + texture = textured geometry

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

Mapping Textures to Surfaces

"Putting the image on the surface"

- Need a function *f* that tells where each point on the image goes
- Similar to parametric surface function
- For parametric surfaces you get *f* for free



Texture Coordinate Functions

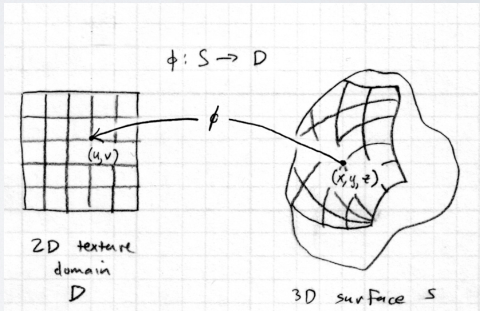
Non-parametrically defined surfaces: more to do

- Can't assign texture coordinates as we generate the surface
- Need ***inverse*** of the function ***f***

Texture coordinate fn.

$$\phi : S \to \mathbb{R}^2$$

- For a vertex at **p** get texture at $\phi(\mathbf{p})$



Texture Coordinate Functions

Define texture image as a function

$$T : D \to C$$

- Where ***C*** is the set of colors for the diffuse component

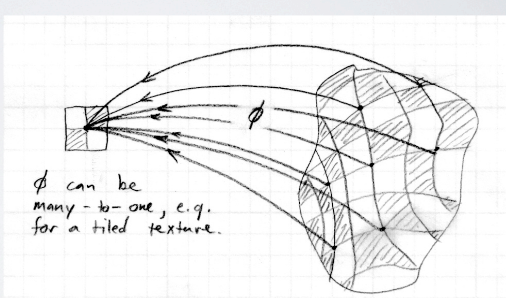
Diffuse color (for example) at point **p** is then

$$k_D(\mathbf{p}) = T(\phi(\mathbf{p}))$$

Texture Coordinate Functions

Mapping from ***S*** to ***D*** can be many-to-one

- Every surface point gets only one color assigned
- OK (and in fact useful) for multiple surface points to be mapped to the same texture point
  - e.g. repeating tiles



Repeating Textures

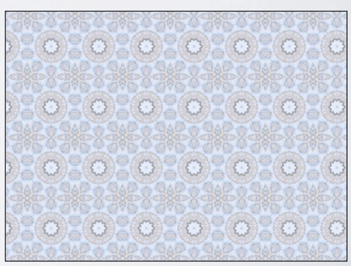
Image Tiles allow repeating textures

- Images must be manipulated to allow tilling
- Often result in visible artifacts
  - There are methods to get around artifacts....

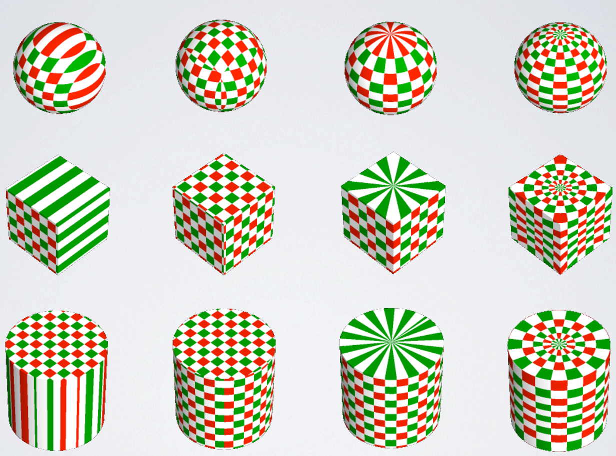
Repeating Textures

Image Tiles allow repeating textures

- Images must be manipulated to allow tilling
- Often result in visible artifacts
  - Artifacts not an issue for some artificial textures



Examples

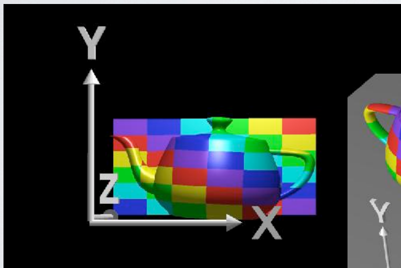


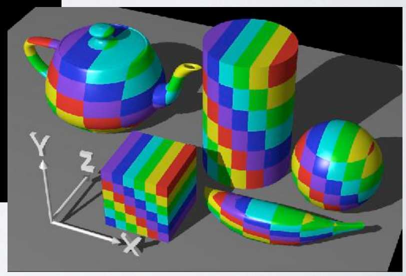
[Paul Bourke]

Planar mapping

Like projections, drop z coord $(u,v) = (x,y)$

Problems: what happens near $z = 0$?

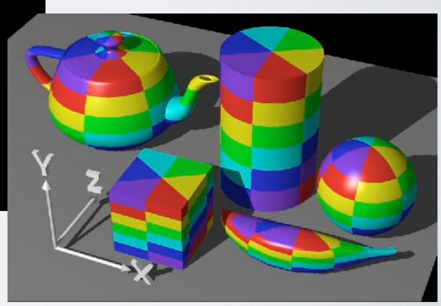




Cylindrical Mapping

Cylinder: r, θ, z with $(u,v) = (\theta/(2\pi), z)$

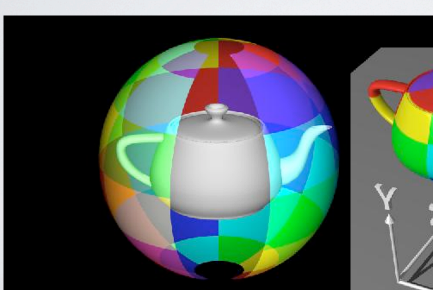
- Note seams when wrapping around ($\theta = 0$ or 2π)

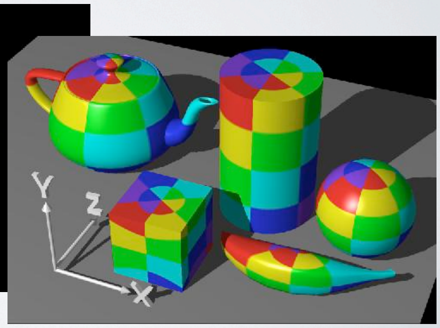


Spherical Mapping

Convert to spherical coordinates: use latitude/long.

- Singularities at north and south poles



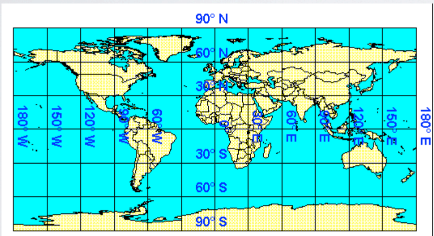


Sphere Mapping

For a sphere: latitude-longitude coordinates

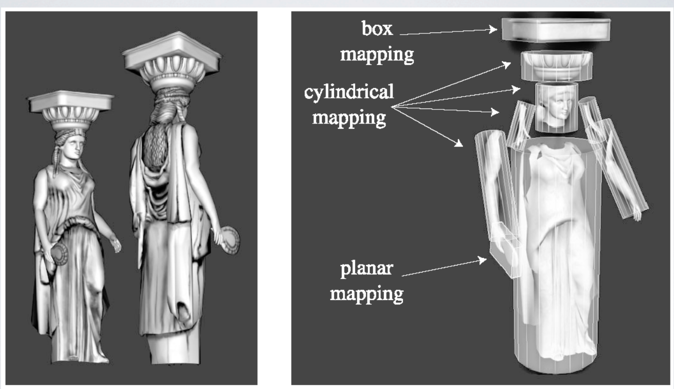
- ϕ maps point to its latitude and longitude

[map: Peter H. Dana]



Examples of Coordinate Functions

Non-parametric surfaces: project to parametric surface

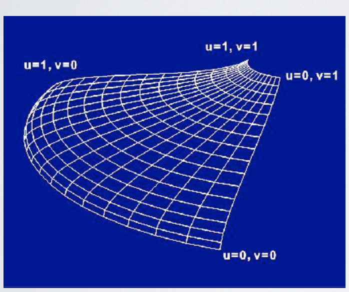


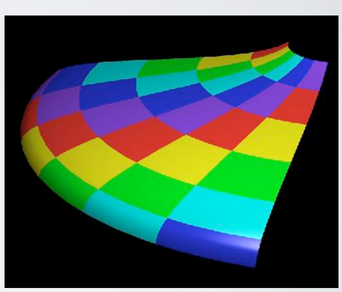
[Moller & Haines 2002]

Examples: Parametric Surface

A parametric surface (e.g. spline patch)

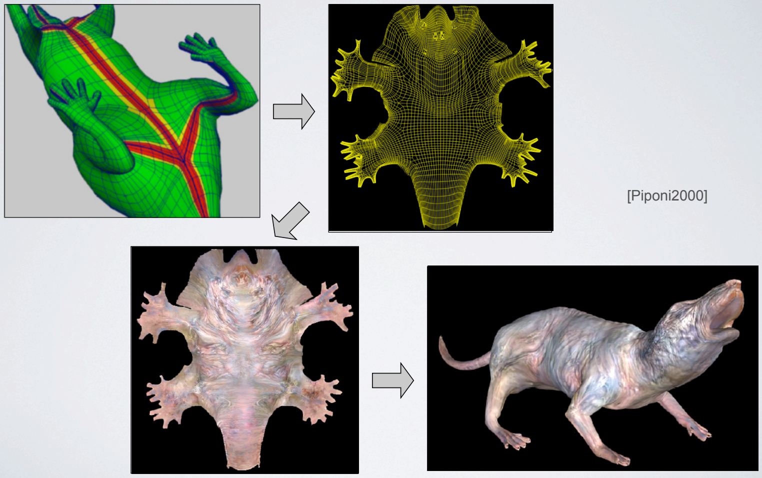
- Surface parameterization gives mapping function directly (well, the inverse of the parameterization)





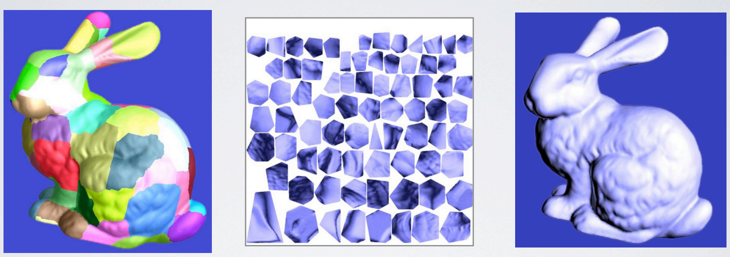
[Wolfe / SG97 Slide set]

Unfold Surface via Optimization



[Piponi2000]

Split Surface into Patches via Opt.



charts atlas surface

[Sander2001]

Texture Coordinates on Meshes

Texture coordinates become per-vertex data

- Can think of them as 2nd position
  - Each vertex has a position in 3D space and in 2D texture space

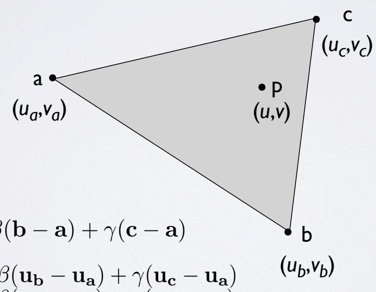
How to come up with vertex (*u*,*v*)s?

- Use any or all of the methods just discussed
  - In practice this is how you implement those for curved surfaces approximated with triangles
- Alternatively: Use some kind of optimization
  - Try to choose vertex (*u*,*v*)s to result in a smooth, low distortion map

Examples: Triangle

Triangles

- specify (*u*,*v*) for each vertex
- define (*u*,*v*) for interior by barycentric (linear) interpolation



$$\mathbf{p}(\beta,\gamma) = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a})$$

$$\mathbf{u}(\beta,\gamma) = \mathbf{u_a} + \beta(\mathbf{u_b}-\mathbf{u_a}) + \gamma(\mathbf{u_c}-\mathbf{u_a})$$
$$\mathbf{v}(\beta,\gamma) = \mathbf{v_a} + \beta(\mathbf{v_b}-\mathbf{v_a}) + \gamma(\mathbf{v_c}-\mathbf{v_a})$$

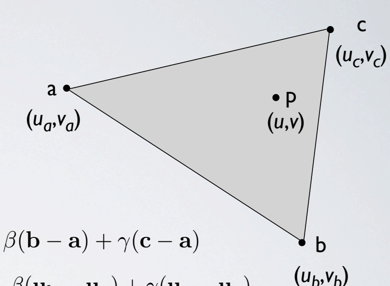
Dealing with Distortion

Can't Linearly Interpolate Tex. Coords.

Triangle Scan Conversion

For each pixel:

1. Interpolate u & v down edges, across spans
2. Look up nearest texel in texture map
3. Color pixel according to texel color
   (possibly modulated by lighting calculations)



$$\mathbf{p}(\beta,\gamma) = \mathbf{a} + \beta(\mathbf{b}-\mathbf{a}) + \gamma(\mathbf{c}-\mathbf{a})$$

$$\mathbf{u}(\beta,\gamma) = \mathbf{u_a} + \beta(\mathbf{u_b}-\mathbf{u_a}) + \gamma(\mathbf{u_c}-\mathbf{u_a})$$
$$\mathbf{v}(\beta,\gamma) = \mathbf{v_a} + \beta(\mathbf{v_b}-\mathbf{v_a}) + \gamma(\mathbf{v_c}-\mathbf{v_a})$$

Leads to Artifacts: http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide05.html

Naïve Texturing Artifacts

Warping at edges of triangles

A more obvious example:

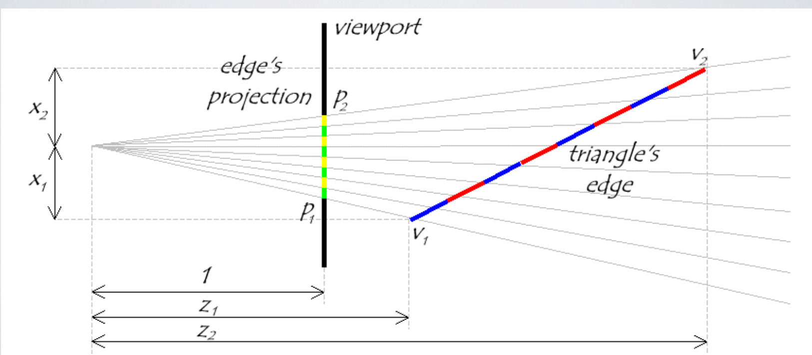
http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide06.html

Consider the geometry of interpolating parameters more carefully

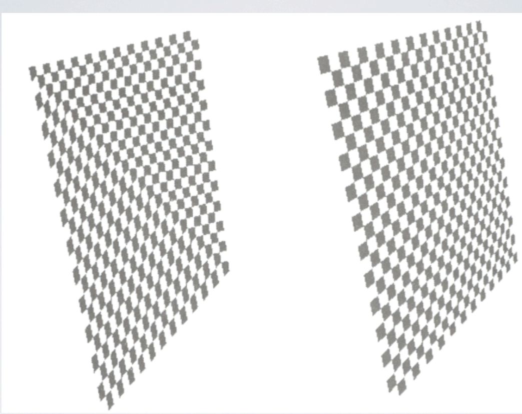
Interpolating Parameters

Problem due 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 not being applied to interpolated parameters

- Parameters should be compressed with distance
- Linearly interpolating them in screen-space doesn't do this

Is this a problem with Gouraud shading?

- A: It can be, but we usually don't notice (***why?***)
  http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide17.html

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 divide per-pixel

Problem: may not know ***z*** (didn't need it for rasterizing)

- Solution: we do know $w \propto z/n$
- So…interpolate ***w, u/w*** and ***v/w*** (interpolation of ***w*** is non-lin. - see book)

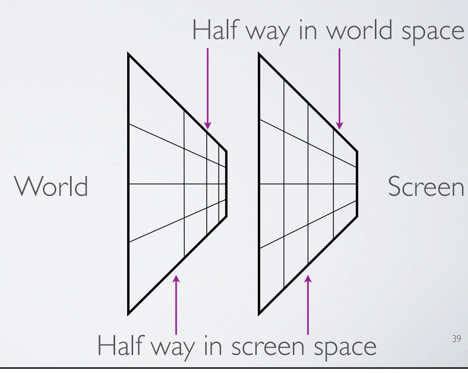
Unfortunately involves a divide per pixel

- http://graphics.lcs.mit.edu/classes/6.837/F98/Lecture21/Slide14.html

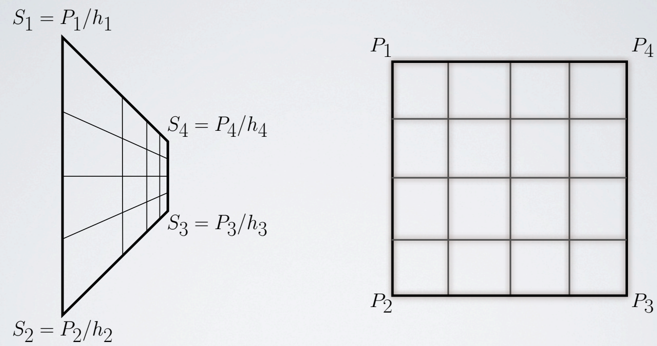
Depth Distortion

Recall depth distortion from perspective

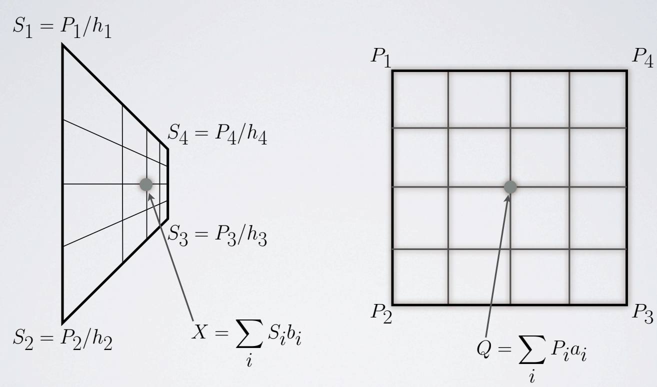
- Interpolating in screen space different than in world
- Ok, for shading (mostly)
- Bad for texture



Depth Distortion

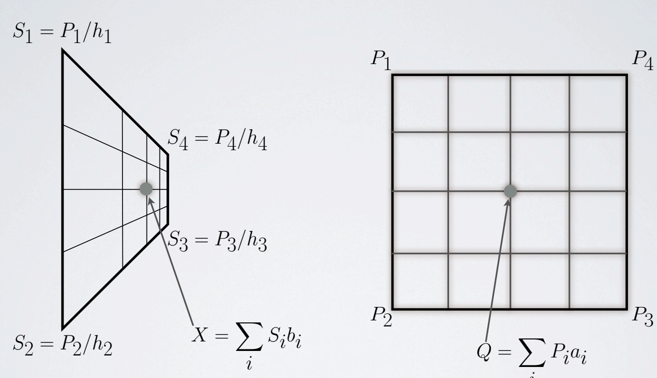


Depth Distortion



We know the s_i , P_i , and b_i , but not the a_i

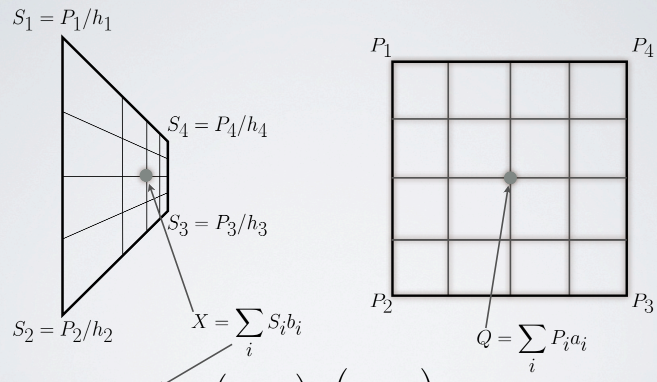
Depth Distortion



$$X = Q/h = \left(\sum_i P_i a_i\right) / \left(\sum_j h_j a_j\right)$$

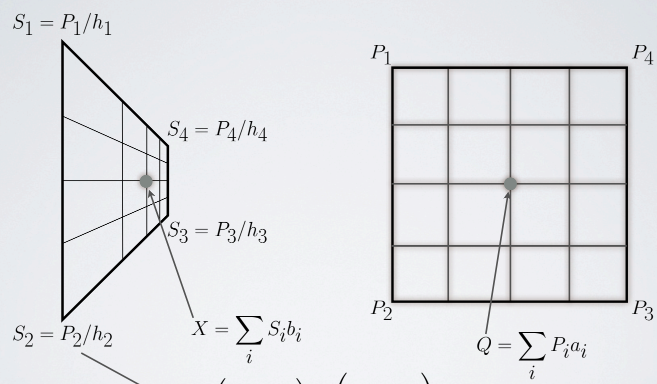
42

Depth Distortion



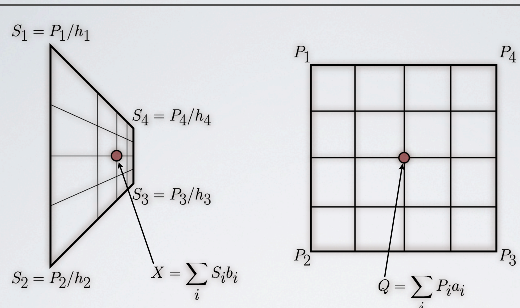
$$\sum_i S_i b_i = \left(\sum_i P_i a_i\right) / \left(\sum_j h_j a_j\right)$$

Depth Distortion



$$\sum_i P_i b_i / h_i = \left(\sum_i P_i a_i\right) / \left(\sum_j h_j a_j\right)$$

Depth Distortion

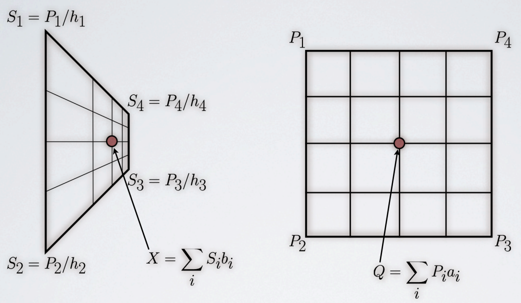


$$\sum_i P_i b_i / h_i = \left(\sum_i P_i a_i\right) / \left(\sum_j h_j a_j\right)$$

Formula for a_i should be independent of given vertex locations P_i

$$b_i / h_i = a_i / \left(\sum_j h_j a_j\right) \quad \forall i$$

Depth Distortion

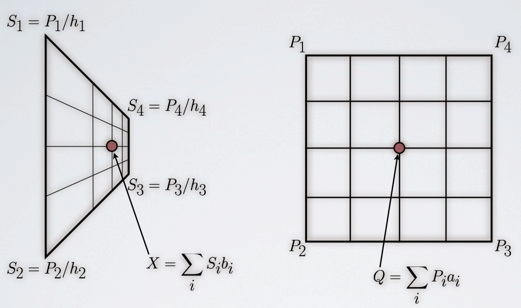


$$b_i/h_i = a_i/\left(\sum_j h_j a_j\right) \quad \forall i$$

Linear equations in the a_i .

$$\left(\sum_j h_j a_j\right) b_i/h_i - a_i = 0 \quad \forall i$$

Depth Distortion



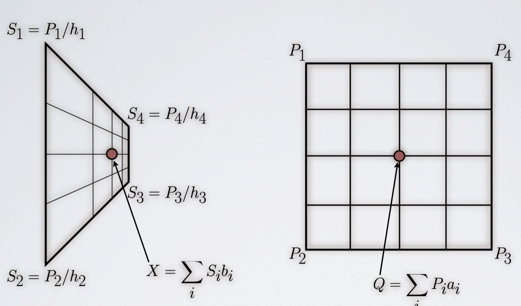
Linear equations in the a_i .

$$\left(\sum_j h_j a_j\right) b_i/h_i - a_i = 0 \quad \forall i$$

Not invertible so add some extra constraints.

$$\sum_i a_i = \sum_i b_i = 1$$

Depth Distortion



For a line: $a_i = h_2 b_i/(b_1 h_2 + h_1 b_2)$

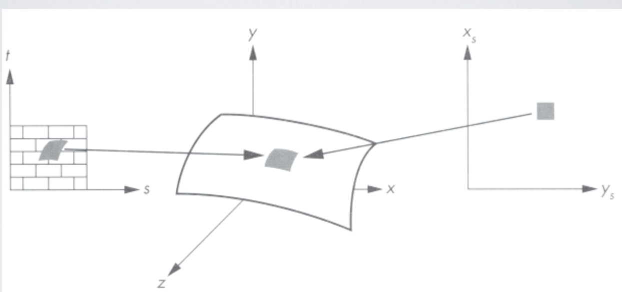
For a triangle: $a_i = h_2 h_3 b_i/(h_2 h_3 b_1 + h_1 h_3 b_2 + h_1 h_2 b_3)$

Obvious Permutations for other coefficients.

Antialiasing

Texture Filtering

Must sample texture to determine color at each pixel in image

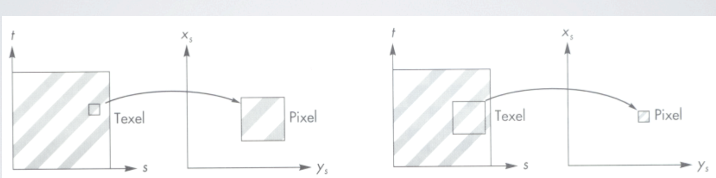


Angel Figure 9.4

Texture Filtering

Size of filter depends on projective warp

- Can prefilter images
  Mip maps
  Summed area tables



Magnification

Minification

Angel Figure 9.14

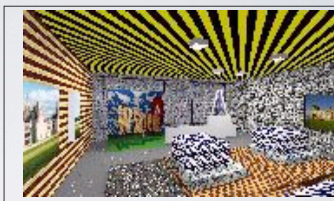
Mip Maps

Keep textures prefiltered at multiple resolutions

- For each pixel, linearly interpolate between two closest levels (e.g., trilinear filtering)
- Fast, easy for hardware

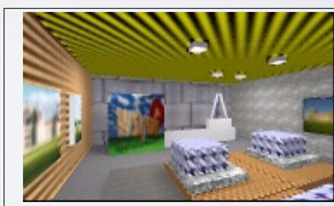
MIP-map Example

No filtering:



AAAAAAAGH
MY EYES ARE BURNING

MIP-map texturing:

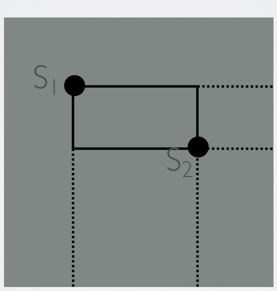


Where are my glasses?

Summed-area tables

At each texel keep sum of all values down & right

- To compute sum of all values within a rectangle, simply subtract two entries
- Better ability to capture very oblique projections
- But, cannot store values in a single byte



Summed-Area Tables

Mipmaps assume each pixel projects to a square in texture (is a lie)

SAT can integrate texels covered by the pixel more exactly (still quickly)

MIP-map texturing

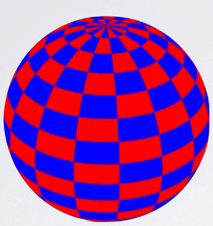
Summed-area table texturing

Uses of Textures

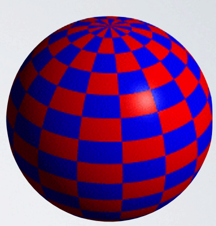
Texture Mapping Variations

Can modulate any parameter in the Phong shading equation

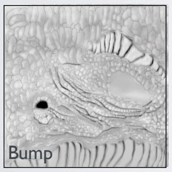
Texture as R,G,B:



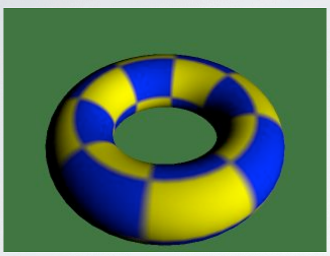
Texture as diffuse lighting coefficients:



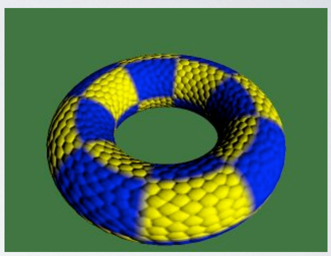
Non-Color Textures



Bump Mapping



No bump mapping

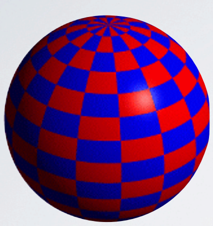


With bump mapping

Images by Paul Baker
www.paulsprojects.net

Bump Mapping

Texture = change in surface normal!

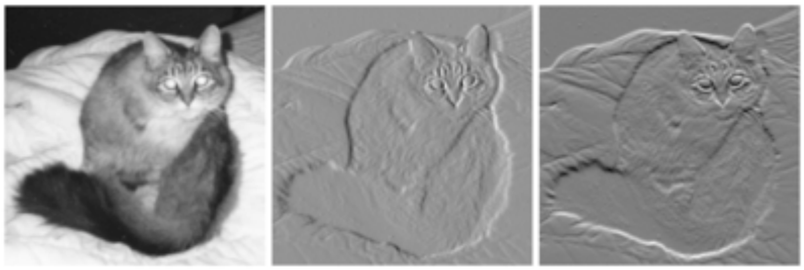


Sphere w/ diffuse texture

Swirly bump map

Sphere w/ diffuse texture and swirly bump map

Image Gradients as Bump Maps

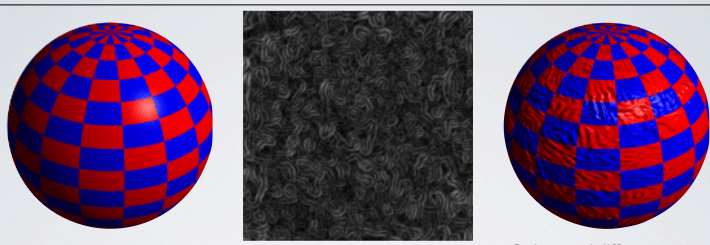


Intensity Image | Horizontal Gradient | Vertical Gradient

Modify surface normal using gradients of texture image

$$\nabla I(u,v) = \left(\frac{\partial I(u,v)}{\partial u}, \frac{\partial I(u,v)}{\partial v} \right) = (I_u, I_v)$$

Bump Mapping



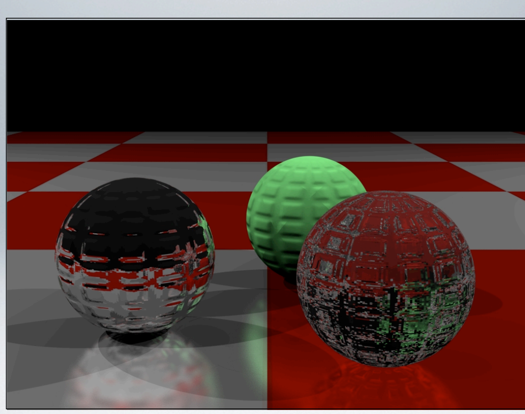
Sphere w/ diffuse texture | *Magnitude Image Gradient* | *Sphere w/ diffuse texture and swirly bump map*

Bump map: $I(u,v)$ contains offsets to normal for each (u,v) pt.

Orig. surface pt, normal and param: $\mathbf{p}, \mathbf{n}, \mathbf{S}(u,v) \quad \mathbf{n} = \mathbf{S}_u \times \mathbf{S}_v$

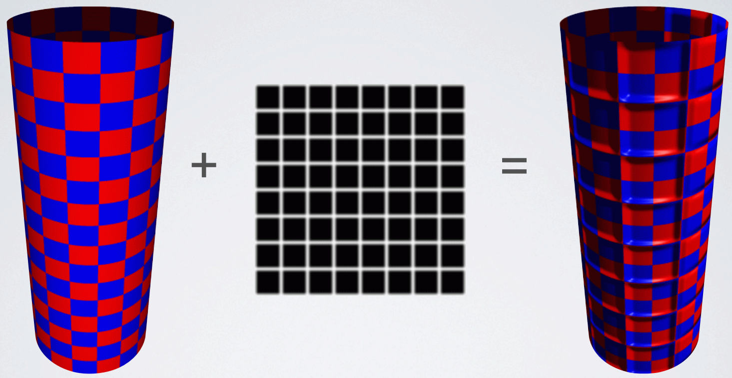
New surface pt and normal: normal: $\mathbf{p}' = \mathbf{p} + I(u,v)\frac{\mathbf{n}}{|\mathbf{n}|}$

$$\mathbf{n}' = \mathbf{n} + \frac{I_u(\mathbf{n} \times \mathbf{S}_v)}{|\mathbf{n}|} + \frac{I_v(\mathbf{n} \times \mathbf{S}_u)}{|\mathbf{n}|}$$



Catherine Bendebury and Jonathan Michaels
CS 184 Spring 2005

More Bump Mapping



How can you tell a bumped-mapped object from an object in which the geometry is explicitly modeled?

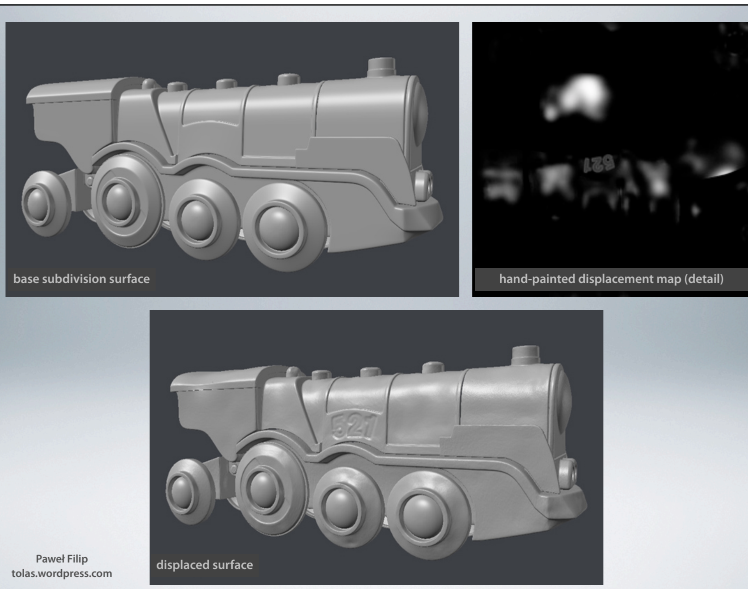
Displacement Maps

Move geometry based on texture map

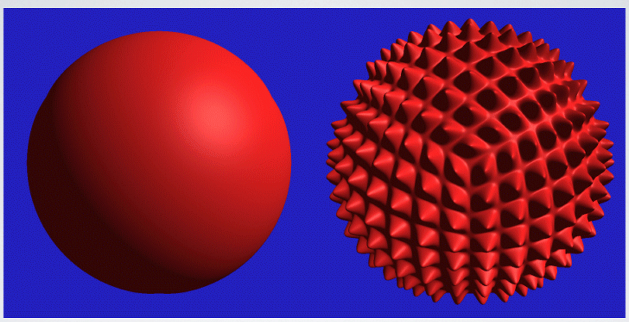
- Expensive and difficult to implement in many rendering systems
- Note silhouette

Bump

Displacement



Displacement Mapping



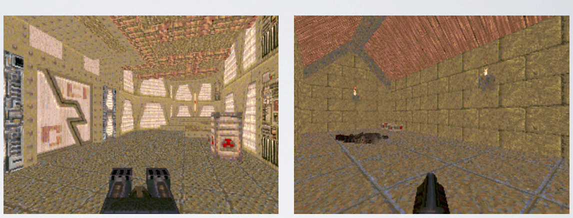
Illumination Maps

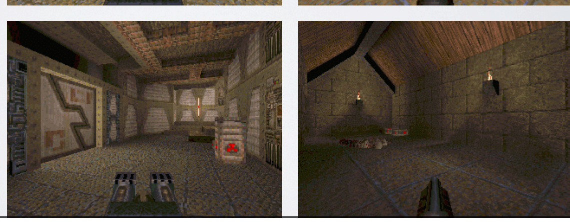
Quake introduced ***illumination maps*** or ***light maps*** to capture lighting effects in video games

Texture map:

Light map

Texture map + light map:





Environment Maps

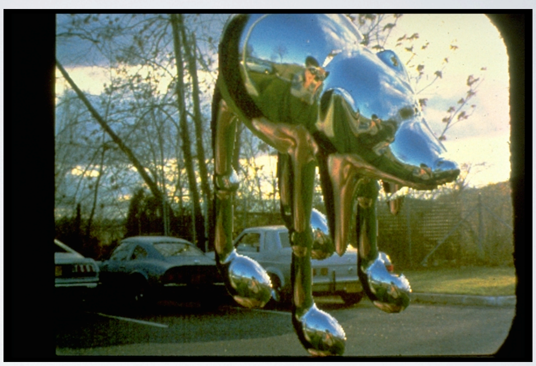
Environment maps allow crude reflections

Treat object as infinitesimal

- Reflection only based on surface normal

Errors hard to notice for non-flat objects

Environment Maps

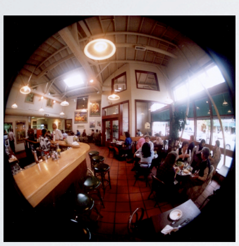


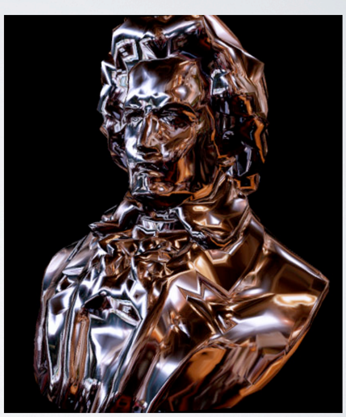
Images from *Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments*
Gene Miller and C. Robert Hoffman
SIGGRAPH 1984 “Advanced Computer Graphics Animation” Course Notes

Environment Maps

Sphere based parameterization

- Wide angle image or
- Photo of a silver ball





Images by Paul Haeberli

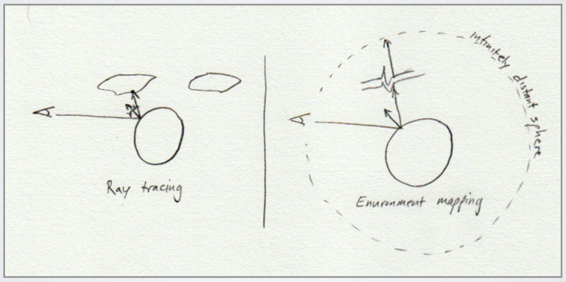
Environment Mapping

From ray tracing we know what we'd like to compute

- Trace a recursive ray into the scene—too expensive

If scene is infinitely far away, depends only on direction

- A two-dimensional function

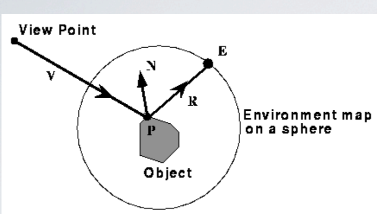


Environment Map

A function from the sphere to colors, stored as a texture

[Blinn & Newell 1976]

Spherical Environment Map

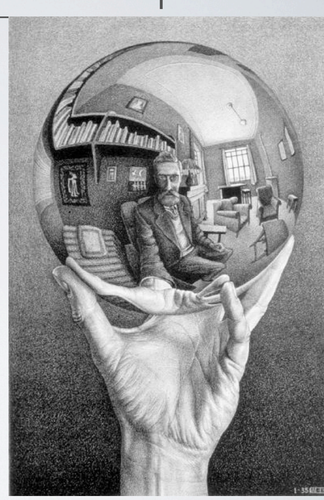


$$m = 2\sqrt{R_x^2 + R_y^2 + (R_z + 1)^2}$$

$$u = \frac{R_x}{m} + \frac{1}{2}$$

$$v = \frac{R_y}{m} + \frac{1}{2}$$

Convert reflection ray direction to point in map



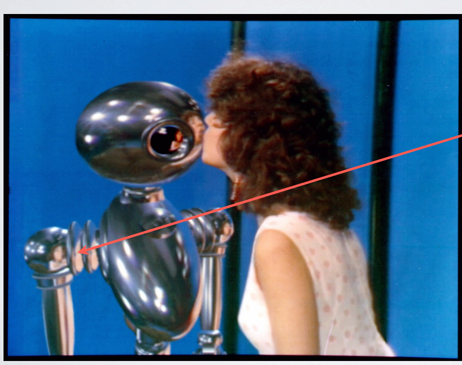
Hand with Reflecting Sphere. M. C. Escher, 1935. lithograph

Environment Maps

Cube based parameterization (see book)

Environment Maps

- Used in 1985 in movie *Interface*
  - Effect by group from the New York Institute of Technology



Note errors

Environment Maps

Used in 1985 in movie *Interface*

- Effect by group from the New York Institute of Technology

Shadow Maps

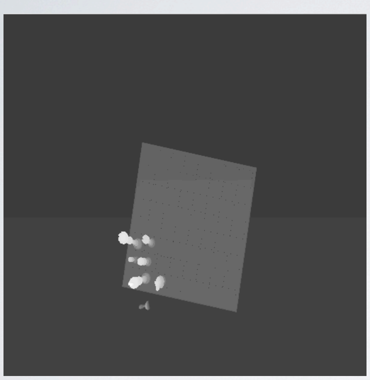
Pre-render scene from perspective of light source

- Only render Z-Buffer (the shadow buffer)

Render scene from camera perspective

- Compare with shadow buffer
- If nearer light, if further shadow

Shadow Maps



Shadow Buffer

Image w/ Shadows

From Stamminger and Drettakis
SIGGRAPH 2002

Note: These images don't really go together, see the paper...

Deep Shadow Maps

Some objects only partially occlude light

- A single shadow value will not work
- Similar to transparency in Z-Buffer

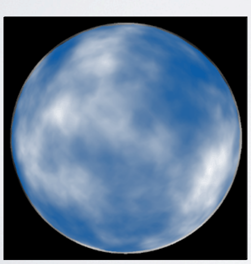
From Lokovic and Veach
SIGGRAPH 2000

Procedural Textures

Procedural Textures

Generate texture based on some function

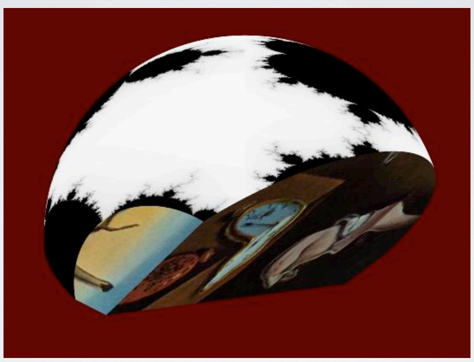
- Well suited for "random" textures
- Often modulate some noise function



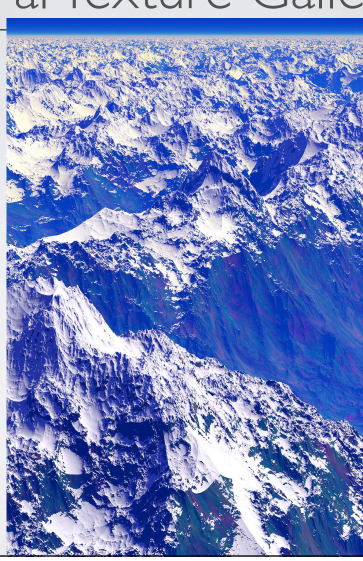
Solid Textures

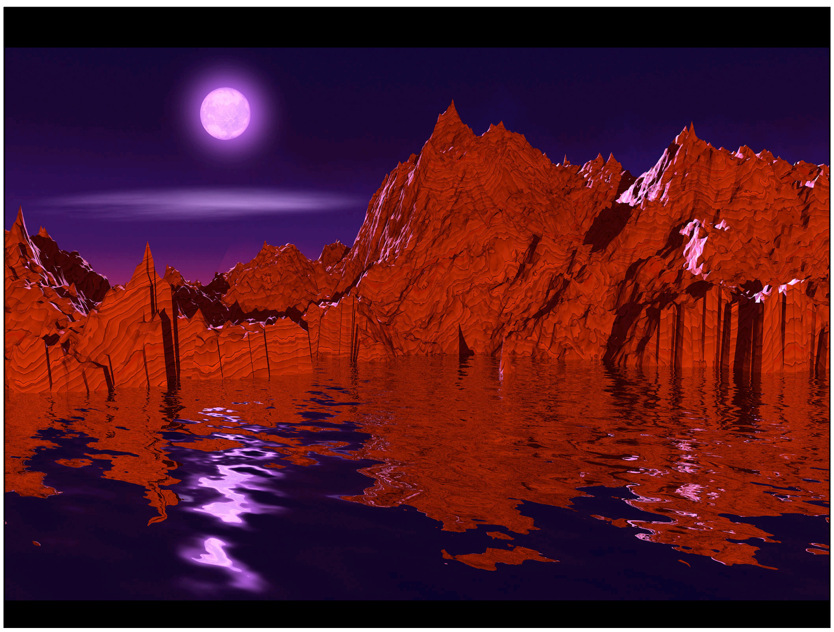
- Texture values indexed by 3D location
  - Expensive storage
  - Compute on the fly, e.g. Perlin noise

Procedural Texture



Procedural Texture Gallery





Metallichron - Brassy Rise
by Armands Auseklis