Texture Mapping Techniques

CSE606: Computer Graphics

Jaya Sreevalsan Nair, IIIT Bangalore

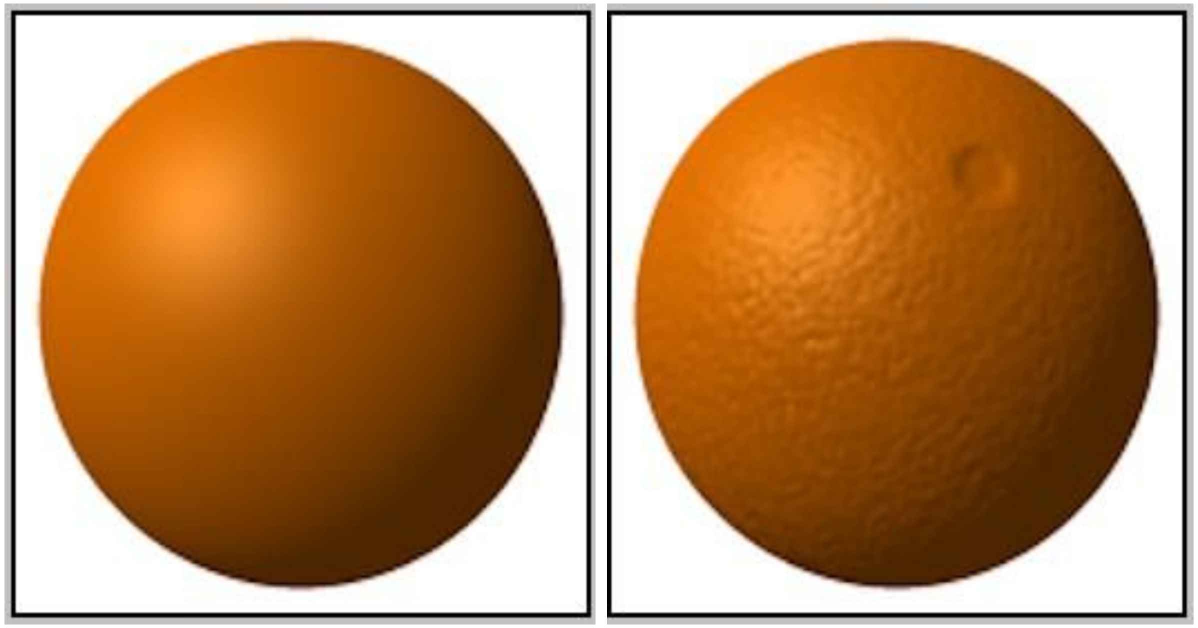
March 26, 2025

Bump Mapping

Introduction

We would like to render “natural” surfaces – typically not smooth, but have small variations over the surface (e.g. orange peel)

Modelling these “bumps” or “dimples” accurately would be very expensive


Sphere with [L] normal shading and [R] desired shading.

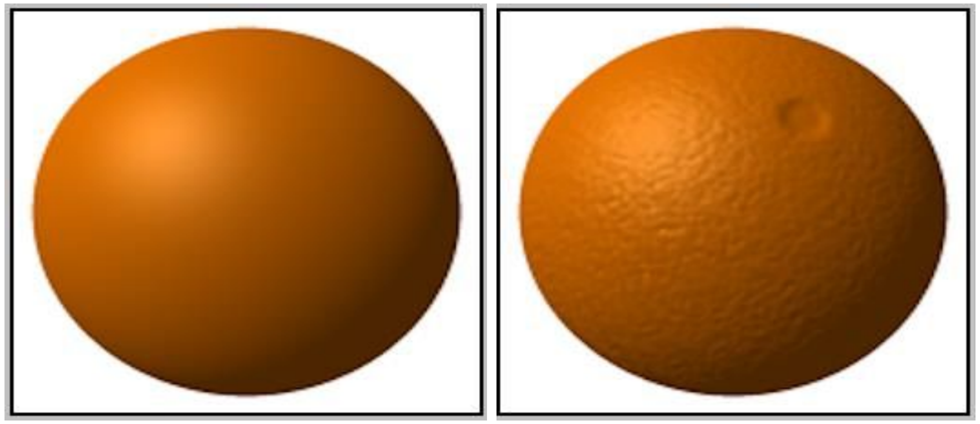
https://en.wikipedia.org/wiki/Bump_mapping

Introduction

We can use textures (e.g. a photograph of the object) to render on a smooth surface.

- But it does not work well if the view direction is changed.
- We get shades of the dimples rather than their correct orientation.

Ideally, we should perturb the normal at each point on the surface and compute the appropriate colour at that point


Sphere with [L] normal shading and [R] desired shading.

https://en.wikipedia.org/wiki/Bump_mapping

Bump Mapping

Given tangent vectors $\mathbf{p}_u$, $\mathbf{p}_v$, the normal $\mathbf{n} = \mathbf{p}_u \times \mathbf{p}_v$. We would like to perturb the normal direction – by adding a small vector in the tangent plane of the surface

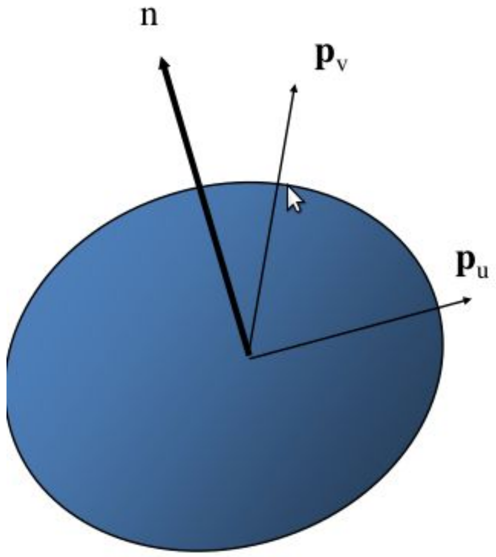
Let d(u,v) be the bump or displacement function, |d(u,v)| << 1

Then, we can compute perturbed normal as: $\mathbf{n'} \approx \mathbf{n} + \frac{\partial d}{\partial u} \cdot \mathbf{n} \times \mathbf{p}_v + \frac{\partial d}{\partial v} \cdot \mathbf{n} \times \mathbf{p}_u$

The vectors $\mathbf{n}\mathrm{x}\mathbf{p}_v$ and $\mathbf{n}\mathrm{x}\mathbf{p}_u$ lie in the tangent plane, hence the normal is displaced in the tangent plane.

The partial derivatives can be precomputed.

The shading can be computed using modified normals.



Implementation

Suppose we use function d(u,v) and sample it to form an array D = [d_{ij}]

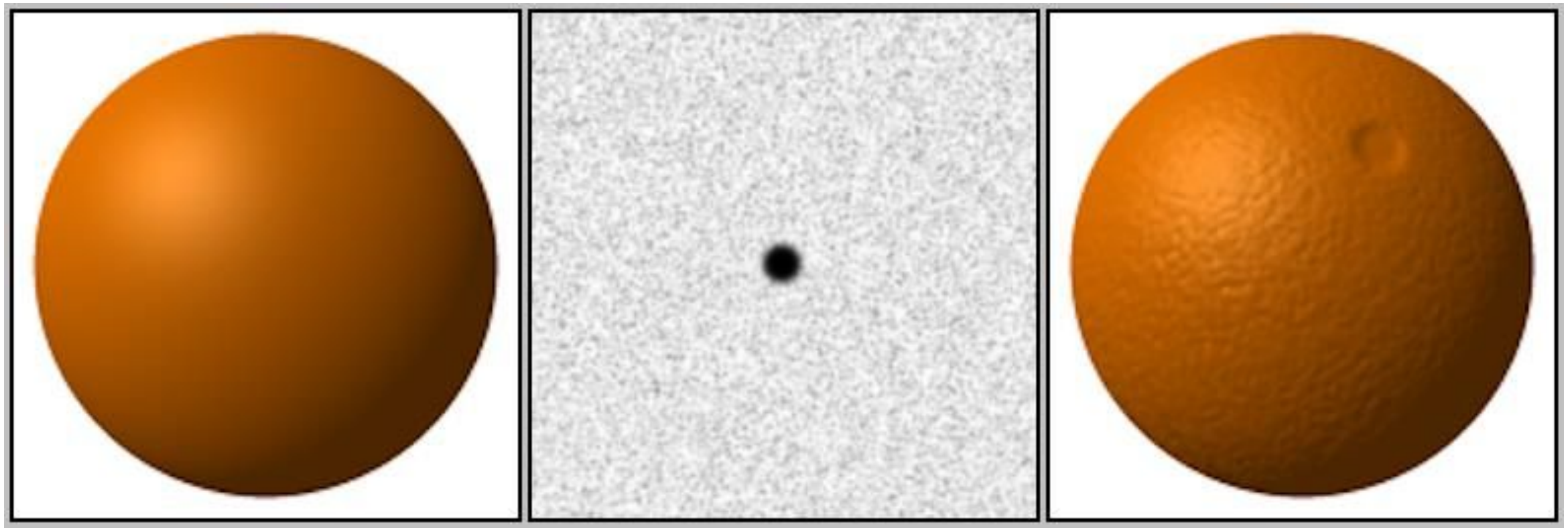
Then
$$\frac{\partial d}{\partial u} \approx d_{(i,j)} - d_{(i-1,j)}$$
$$\frac{\partial d}{\partial v} \approx d_{(i,j)} - d_{(i,j-1)}$$

We can use these computations during shading computation.

- Needs to be applied at all points on the surface – i.e. to every fragment
- Cannot be solved by vertex lighting, or using the fixed function pipeline
- Can be done in the fragment shader

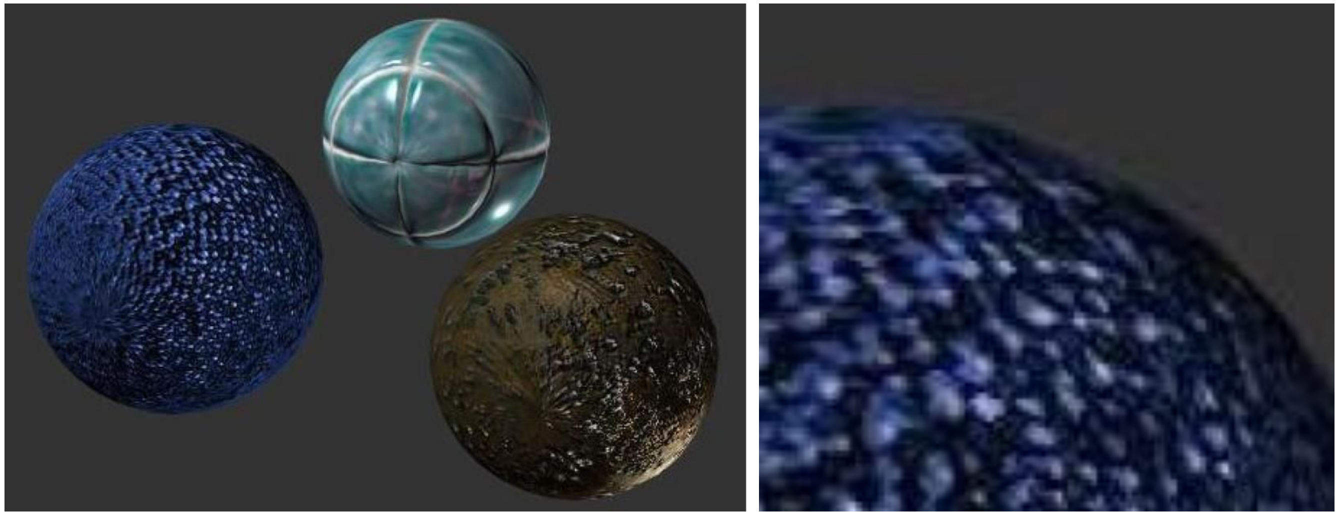
Example

(Left-to-right) Sphere with normal shading, bump map, sphere shaded with bump map.



Features of Bump Maps

- Achieves image complexity without increasing geometry complexity
- Shows variations in shading as viewpoint or lighting changes
- Needs to be implemented using programmable shader (fragment shading) to get real-time performance
- Surface appears bumpy, but silhouette is smooth, which has to be addressed separately



http://www.siggraph.org/education/materials/HyperGraph/mapping/bumpmap.htm

Environment Mapping

Introduction

- Rendering of highly reflective surfaces not handled well by normal rendering schemes
    - Use only local information
    - Should show specular reflections that mirror the environment
- Ray tracing schemes accomplish this well, but require global calculations
- Environment maps and reflection maps provide an alternative approach
    - Uses techniques similar to textures to get a visually acceptable result with moderate computing requirement
    - First proposed by Newell and Blinn in 1976

Environment Maps: Approach

http://graphics.ucsd.edu/courses/rendering/2008/kdemesa/

Consider a (reflective) polygon that we are trying to render – model it as a mirror

Given the position of the viewer, a point on the polygon, and the surface normal, the reflected ray can be computed

The intersection of the reflected ray with the environment provides information about the color/ texture to be used for rendering that point of the polygon

Paul Debevec

Sample videos

Paul Debevec: Rendering with Natural Light, SIGGRAPH 98 Electronic Theater

https://www.pauldebevec.com/RNL/

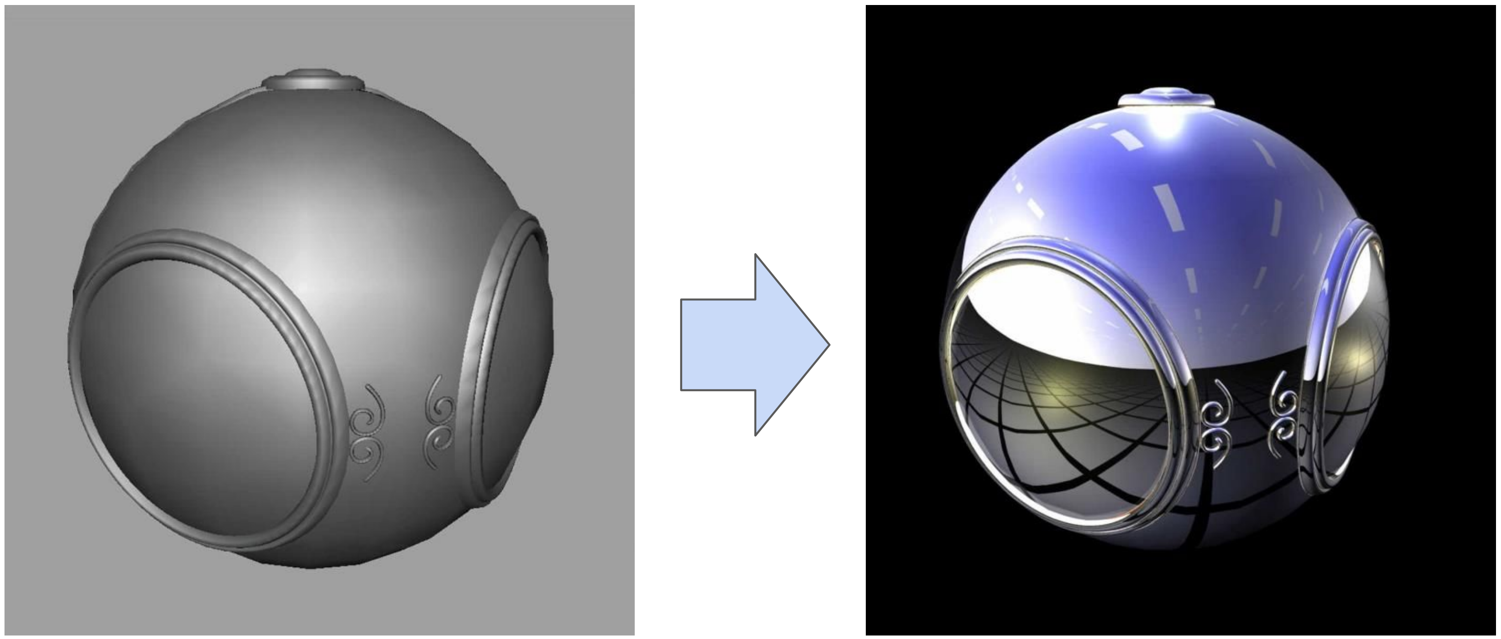
Paul Debevec Fiat Lux SIGGRAPH 99 Electronic Theater

https://www.pauldebevec.com/FiatLux/movie/

Environment Maps: Approach

Compute in two passes:

1. Place a camera at the center of the mirror, pointing towards the surface normal
   a. Image obtained is used as the image of the environment
2. Use this image as texture to render the mirror surface

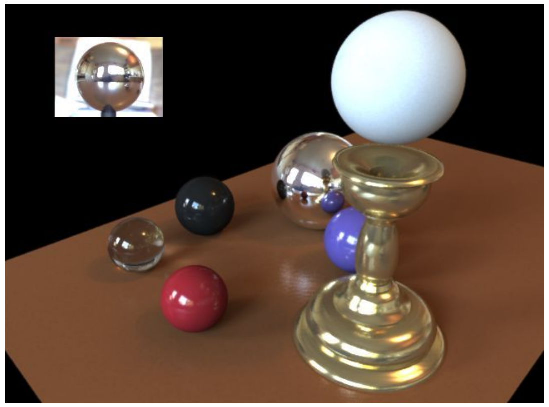


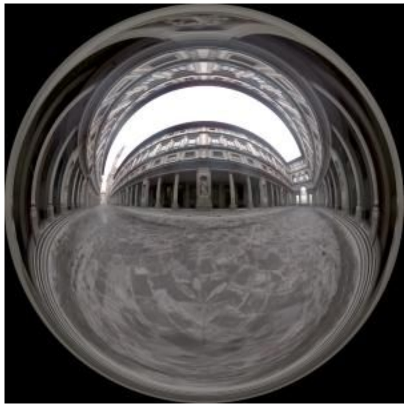
Environment Maps - Light Probes

Objective: Capture the “natural” light falling at a point in a scene.

- Place a shiny spherical object at the desired location.
- Capture one or more high-res, high dynamic range photographs of the shiny object.
- Stitch these together to cover as much of the spherical view as possible. Generate a circular or rectangular “texture” image from this. Use this to map colors onto the surface being rendered - for reflected rays.

Paul E. Debevec. *Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography*. In SIGGRAPH 98, July 1998





Paul Debevoc

Limitations

Key Assumptions:

- Environment objects are far from the object.
  - If not, will have parallax effects – images of environment objects not where they should be
- No self reflections – object is convex.
  - Only renders the environment, not the object itself
- Objects do not reflect each other.
  - Or, objects in the environment are not reflective

Approximations

Environment mapping produces acceptable results in many situations:

- Reflections artifacts, especially on curved surfaces, are often not noticed by the viewer
  - More noticeable on flat surfaces
- As long as reflections on curved surfaces are approximately correct and correlate to curvature changes on the surface, physical accuracy is not needed.

- Ideally, need an environment map for each object being rendered
- Map needs to be recomputed when either the reflective object or objects in the environment move.
- However, in most cases, the same map can be re-used for a range of objects and movement. Effects sufficiently “believable”

Where should the first pass projection be done, so that the image can be used in the second pass?

What should be the position and direction of the camera?

Implementation in OpenGL

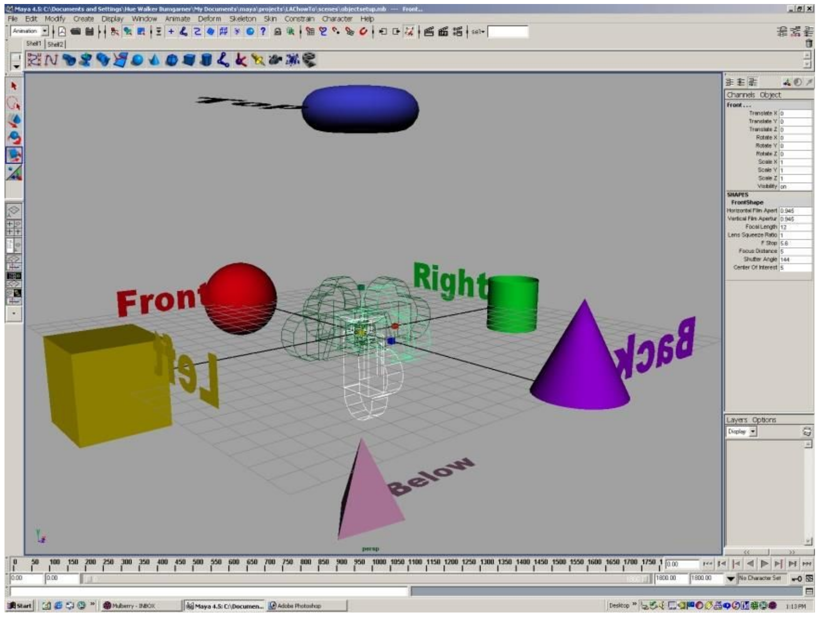
OpenGL/WebGL provides support for environment maps, i.e. spherical (and cube maps)

- Render using normal OpenGL process, with camera in desired position
- Create a circular image from this, as an orthographic map of the image on the sphere
  - Can be approximated by obtaining a perspective projection with a very large angle
- Use this as texture (of type environment map) for rendering the object
- Hardware and software support available

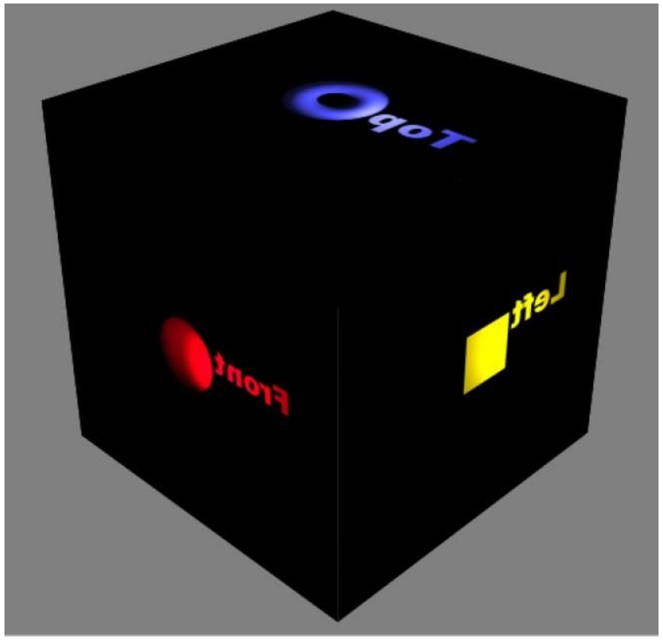
Skyboxes and Cube Maps

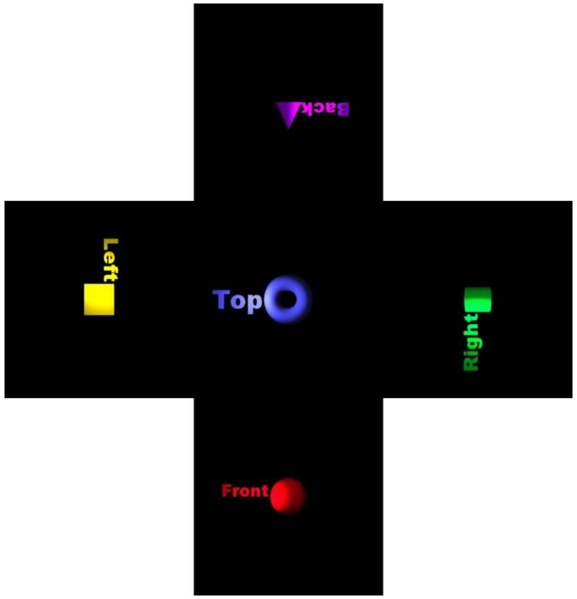
Assume environment is a cube enclosing the scene.

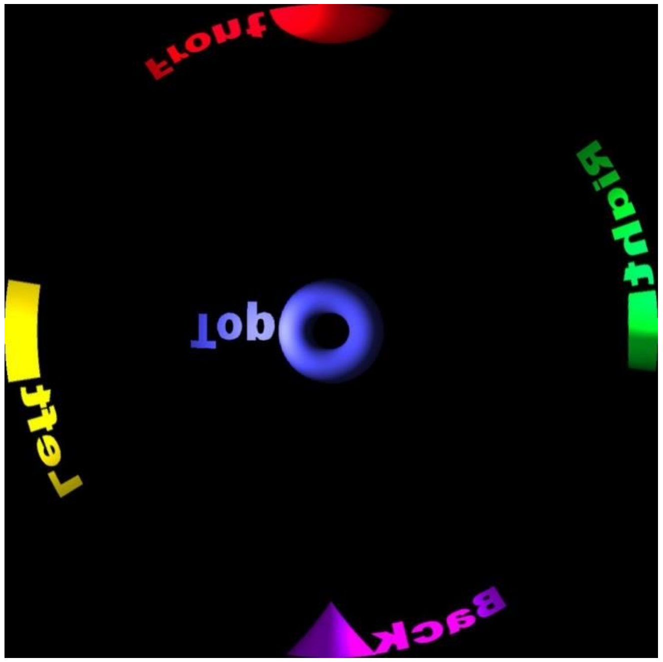
- Natural in scenes such as in a room
- Use 6 cameras, each centered at the object and pointing out towards each face of the cube
- Render view of each camera (onto the corresponding face)
- Similar to Skybox (e.g., used in video games,)



Mapping to Cube and Sphere

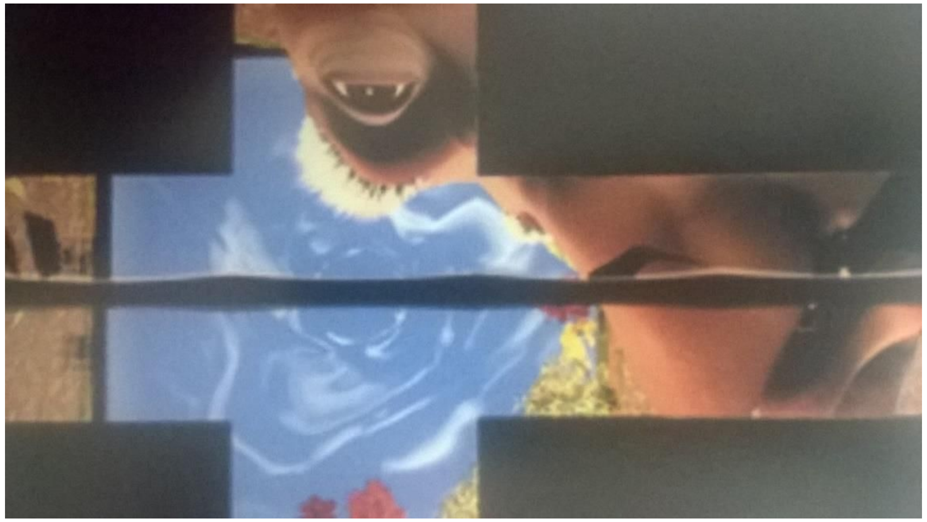






E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

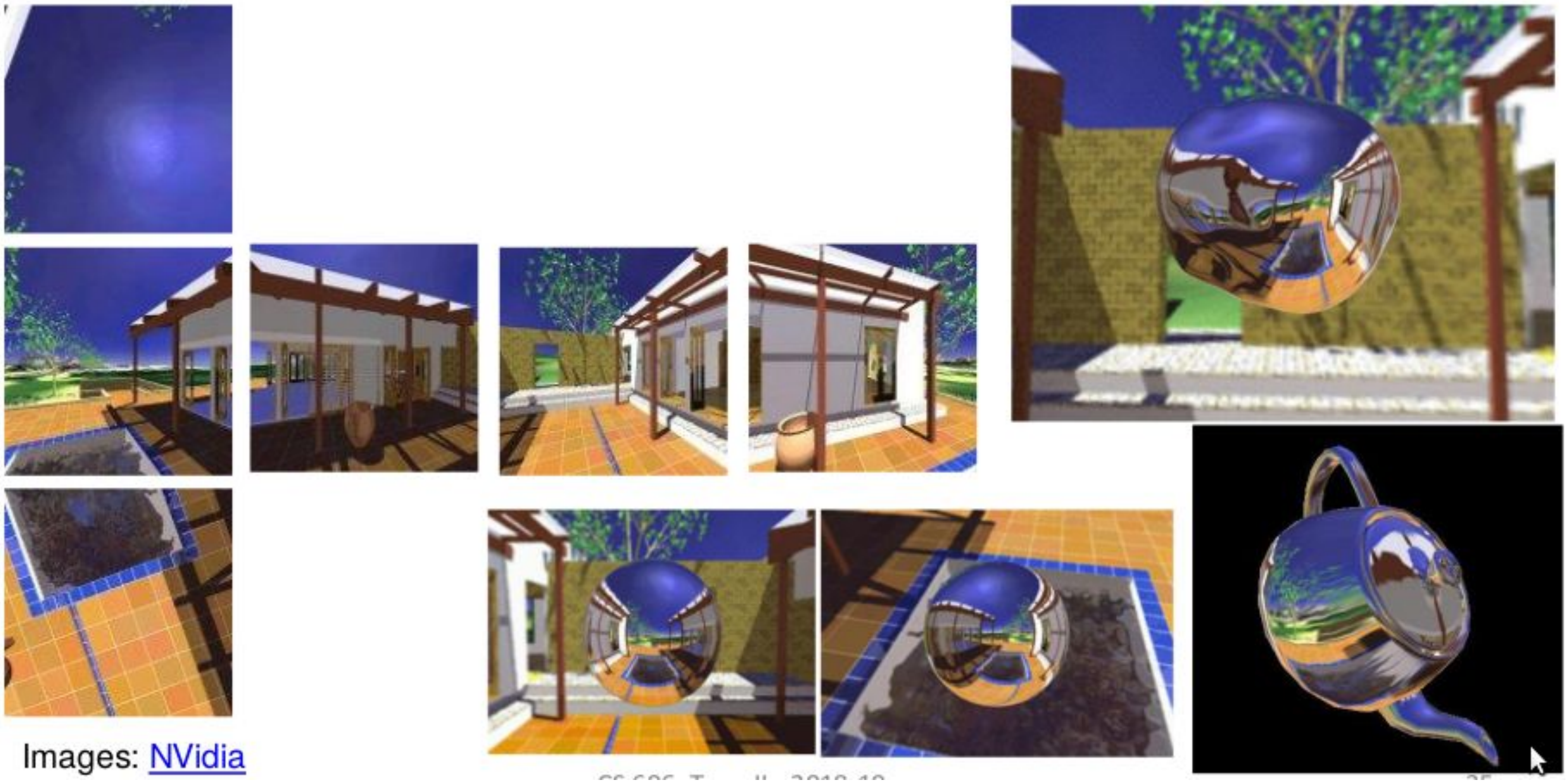
Geri’s Game: Environment Mapping



Reflections/refractions on glass done through environment maps (cube).

Example of Cube Map

http://www.nvidia.com/object/cube_map_ogl_tutorial.html



Images: NVidia

Procedural Mapping

Introduction

Bitmapped textures use textures where pixels correspond to that of an actual image raster.

Bitmapped textures can be realistic when using photographs, however the detail in any region is the same, without any consideration for zooming in.

Hence, procedural textures are used to alleviate this drawback.

http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

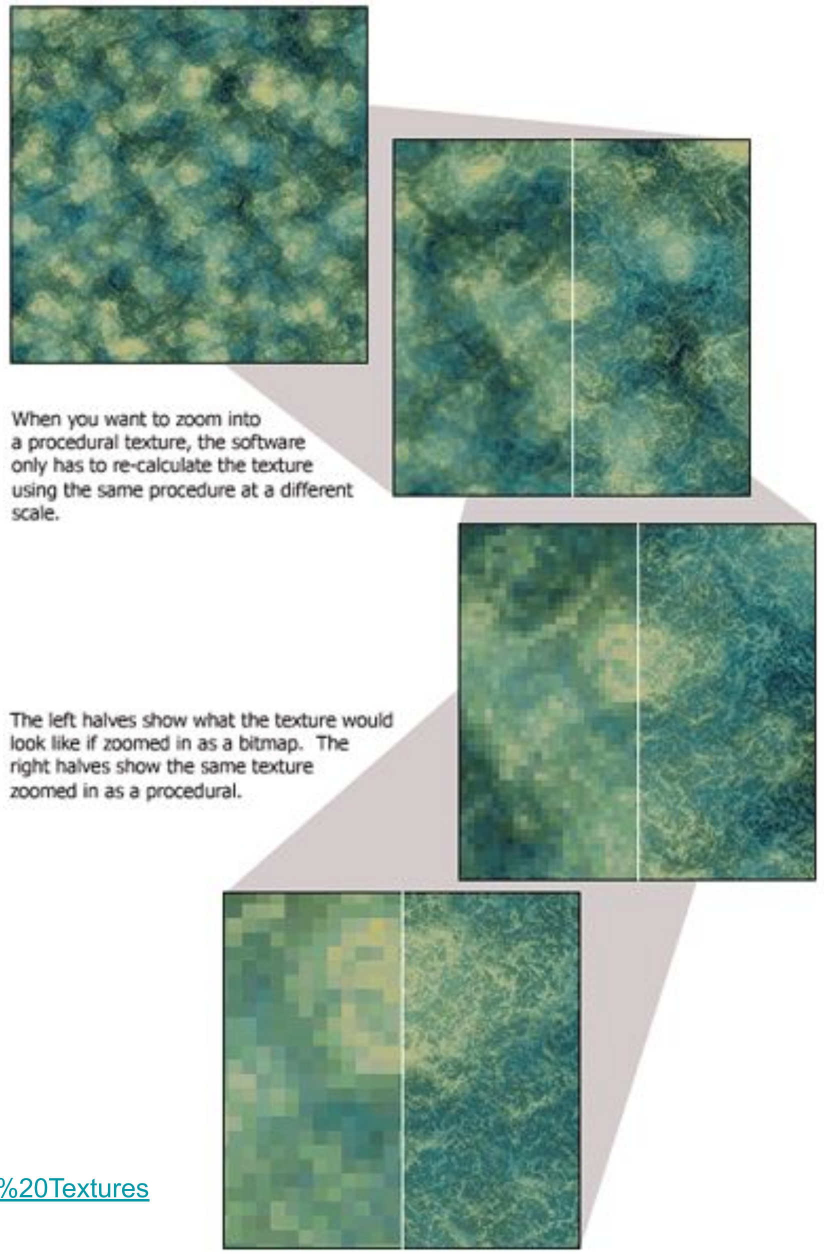
Introduction

Bitmapped textures use textures where pixels correspond to that of an actual image raster.

Bitmapped textures can be realistic when using photographs, however the detail in any region is the same, without any consideration for zooming in.

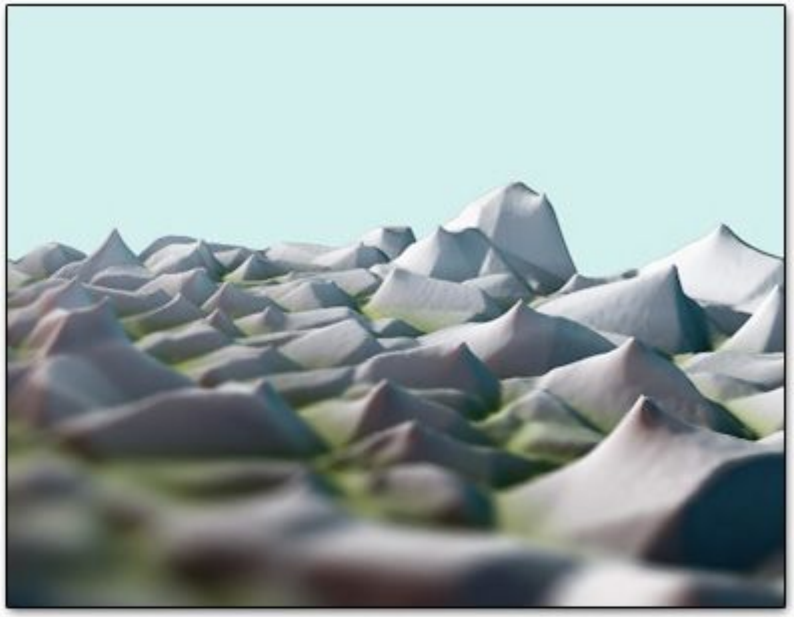
Hence, procedural textures are used to alleviate this drawback.

Procedural textures are created by procedure applied by the parameters.


When you want to zoom into a procedural texture, the software only has to re-calculate the texture using the same procedure at a different scale.
The left halves show what the texture would look like if zoomed in as a bitmap. The right halves show the same texture zoomed in as a procedural.

http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

Examples



http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

Noise Function

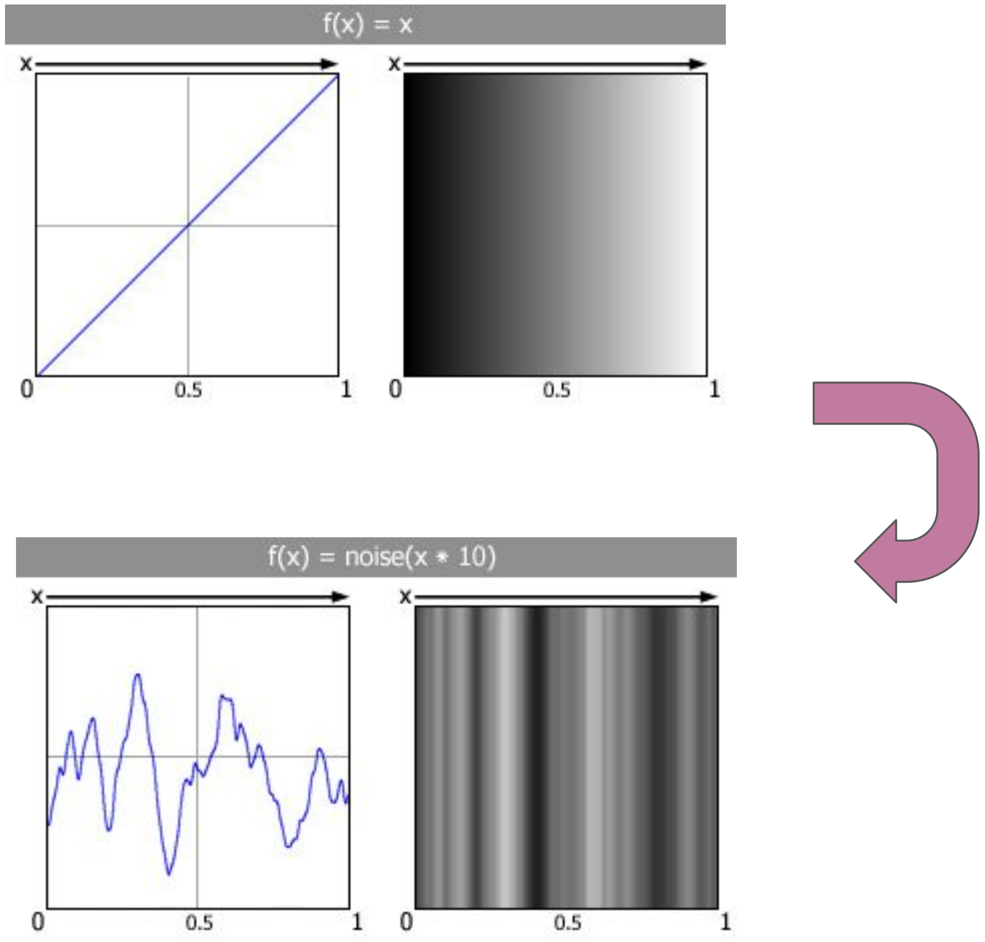
Generally adding a pseudo-random noise function in-place provides procedural textures. The pseudo-random noise was proposed by Ken Perlin in 1983.

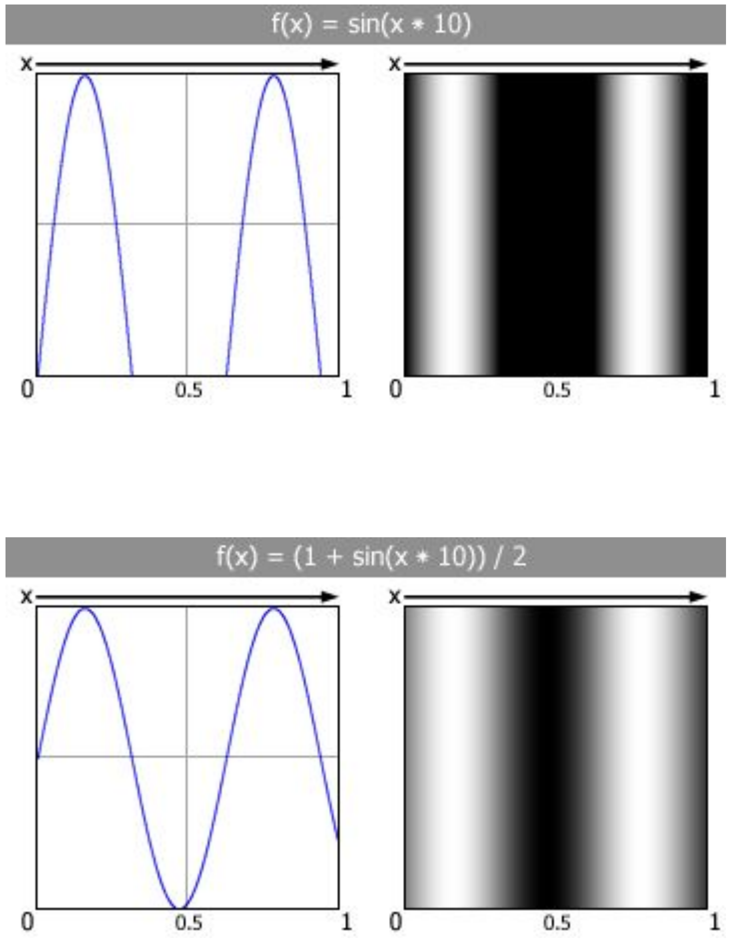
Perlin Noise is given as:

$$f(x, y) = (1 + \sin((x + \text{noise}(x * 5, y * 5) / 2) * 50)) / 2$$

http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

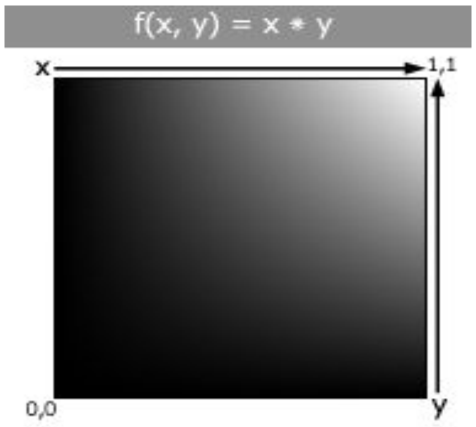
Noise Function

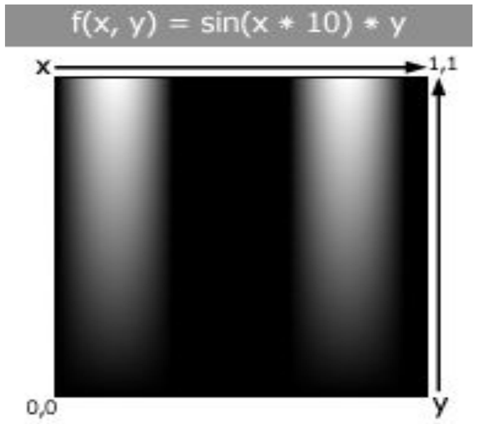


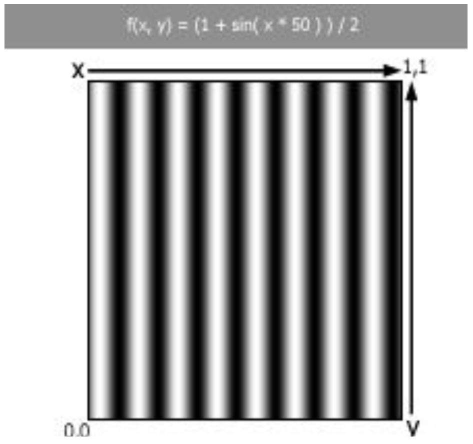


http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

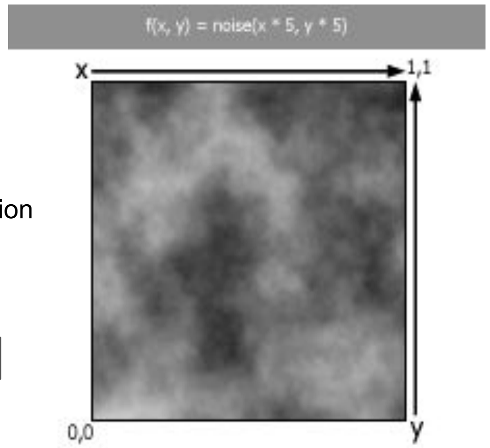
Bivariate Noise Function

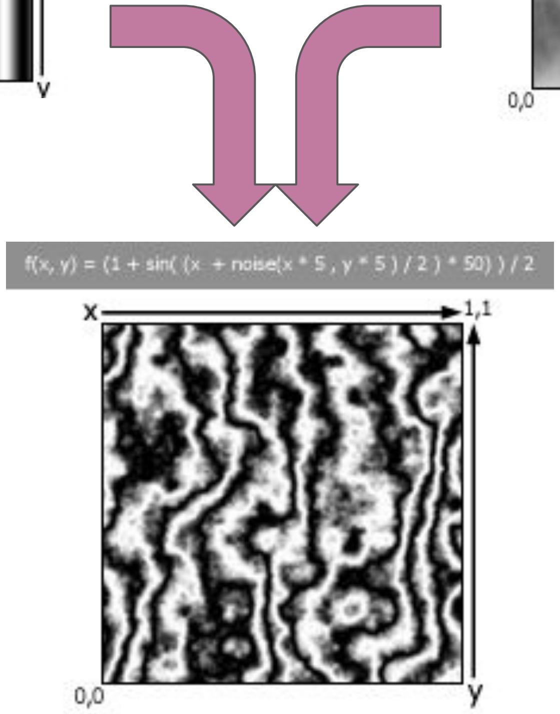






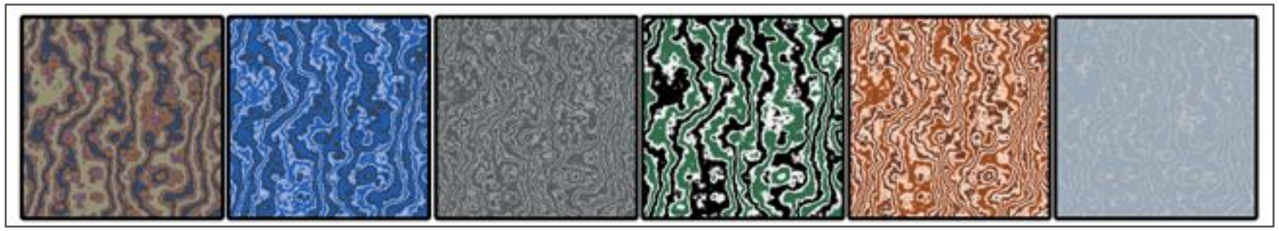
(***Left***) The sin() function alone, with no noise to modify the ***x*** parameter. (***Right***) The noise() function alone.

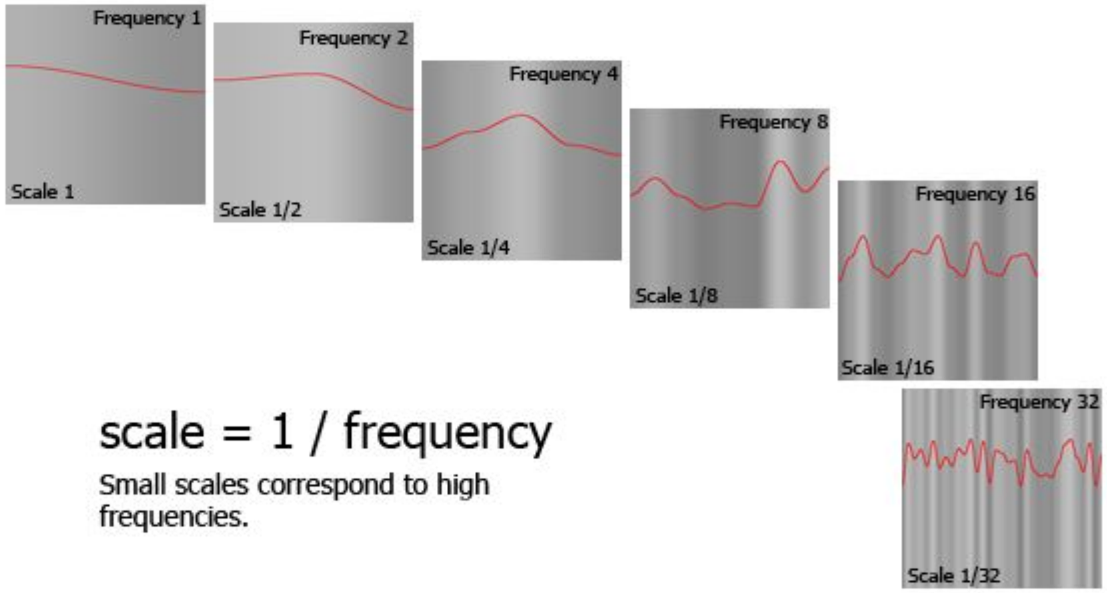




http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

Perlin Noise Textures





scale = 1 / frequency

Small scales correspond to high frequencies.

http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures

Summary

- Bump Maps
- Environment Maps
- Procedural Maps