Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

JOHNS HOPKINS
WHITING SCHOOL
of ENGINEERING

Module 3C
Procedural Texturing

Procedural Textures

- Applying Texture to Ray Tracing
  - 3D and procedural textures
    - Wood grain
    - Perlin noise, turbulence, marble texture
- Texture Mapping

Adding Surface Texture to Ray Tracing

- Two methods of adding textures to ray-traced images
  - Solid or 3D texture
    - Usually a procedural method
    - http://www.upvector.com/?section=Tutorials&subsection=Intro%20to%20Procedural%20Textures
  - Image texture
    - Image is pasted onto the surface of the object
- Use the color returned from the texture function in one of 2 ways:
  - Use the texture color in place of the local illumination
    - “Decal” texture method
  - Use the texture color to modulate the local illumination
    - Modulate just the ambient and diffuse components
    - Modulate the entire local illumination

Three Dimensional Textures

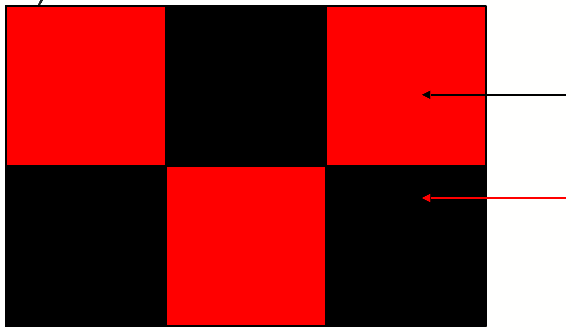
- 3D texture mapping overcomes some of the 2D texture mapping problems
  - Particularly mapping and aliasing issues
    - Mapping 2D textures to complex objects can be difficult
- Solid textures assign texture values at all points in 3D texture space
  - Often called **solid texture**
  - Developed by Perlin and Peachey (1985)
  - Solid textures may have a lattice of texture values within some volume
    - Interpolate between lattice points for intermediate values
  - Most useful for simulating materials such as wood, concrete, marble
    - Internal structure of the material may be important
- Memory costs for storing a high resolution 3D texture is high
  - Procedural methods are generally used
  - Although it eliminates mapping difficulties, texture patterns are limited to definitions that can be analytically constructed
    - 2D texture mapping has a wealth of available frame-grabbed images

Solid Textures

- Procedural texture mapping
  - 3D texture maps are generally not stored because they would be too large
    - Procedurally defined and computed when needed
  - The texture map is defined in three dimensions, T(x,y,z)
  - color = f(x, y, z)
    - Known as the **identity mapping**
    - Color produced depends on position
    - A function (or procedural method) finds the corresponding point in the 3D texture map space
  - Evaluate a 3D texture function at the object's surface points
    - Analogous to sculpting or carving an object out of a block of material (e.g. wood)
    - Object color determined by the intersection of the surface with the texture field
    - For ray traced images, the intersection point is the 3D world point to use in the procedural texture
- Under modeling transformations: perform same transformations to the texture field as the object
  - Or map object into texture field using inverse of modeling transform
    - Such that the texture remains static on the surface of the object
    - However, moving an object through a static texture field can produce interesting effects

Checkerboard

- Simple example: checkerboard
- Boxes of alternating colors
  - Specify box dimensions (C_x, C_y, C_z)
  - Add the integer values of x,y,z divided by their cube dimension
    - Determine if the result is even or odd (mod 2)
      - If even: assign one color, if odd: assign other color
    - ((int)(x/Cx)+(int)(y/Cy)+(int)(z/Cz))%2
  - Issues
    - Truncation to int causes anomalies near 0
      - Can add a constant to translate x,y,z away from 0
    - Can have round off issues with the (int) truncation if a planar surface to be textured lies on a constant integral value (such as z = 0.0)



2.5,1.5=(2+1)%2=1 (odd)

2.8,0.9=(2+0)%2=0 (even)

2D Example with Unit Squares

Wood Grain Texture

- Wood grain can be simulated by a set of concentric cylinders with varying colors
  - As distance from an axis varies color jumps back and forth between 2 alternating values
  - For rings about the z axis: $rings(r) = \left((\text{int})r\right) \% 2$ $\qquad r = \sqrt{x^2 + y^2}$
- Can jump between 2 values: D and D+A

$$wood(\mathrm{x,y,z}) = D + A * rings\left(\frac{r}{M}\right)$$

  - Produces rings of thickness M that are concentric about the z axis
- Can add interest by “wobbling” about the axis
  - Perturb r with a sinusoidal function

$$rings\left(\frac{r}{M} + K\sin\left(\frac{\theta}{N}\right)\right) \qquad \theta = \tan^{-1}\left(\frac{\mathrm{x}}{y}\right)$$

    - Fluctuation of rings as theta varies
    - Makes the ring wobble in and out N times about the z axis
    - K: constant to determine the max variance of the “wobble”

Wood Grain Texture (continued)

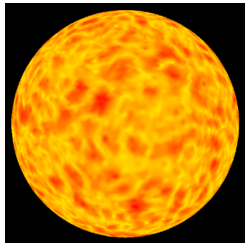
- Add a twist along the axis of the cylinders

$$rings\left(\frac{r}{M} + K\sin\left(\frac{\theta}{N} + Bz\right)\right)$$

- To get textures so it is concentric about other than the z-axis
  - Apply a “tilt” function
  - $r = \sqrt{x'^2 + y'^2}$, where $(x', y', z') = T(x, y, z)$
  - T is a rotational transformation

[Link no longer valid]

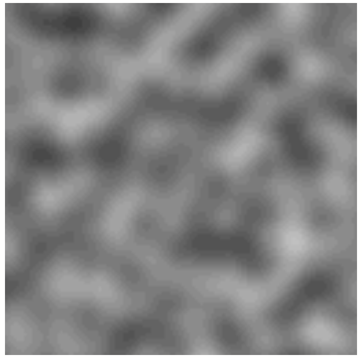
3D Noise Function



- Common basis of solid texture is a pseudo random noise function
- Noise function designed to have the following properties:
  - Statistical invariance under rotation
  - Statistical invariance under translation
  - Narrow bandpass limit in frequency (has no visible features larger or smaller than a certain range)
    - Statistical properties are the same when measured over different areas with different orientations
    - Function can be sampled without aliasing (band limited in frequency)
- Useful to model natural phenomena like clouds, wood, patterns in rock, fire
  - Appearance is random but has recognizable pattern

Perlin's Noise Function

- Perlin (1985) defined a noise(x,y,z) function
  - Returns a single scalar value given a 3D position
- Created by smoothly interpolating between random values
  - In 3D: define integer lattice at locations (i,j,k) where i,j,k are integers
    - Each point on the lattice has a random number associated with it
  - Value of the noise function for points not on the lattice is found by interpolating from nearby lattice points
- Collection of noise values could be stored in a large 3D array
  - Would require lots of storage
- Easy to generate noise value each time it is used
  - float latticeNoise(int i, int j, int k)
  - Function must be efficient and repeatable
    - Always returning the same noise value for a specific (i,j,k)



http://en.wikipedia.org/wiki/Perlin_noise

Lattice Noise Function

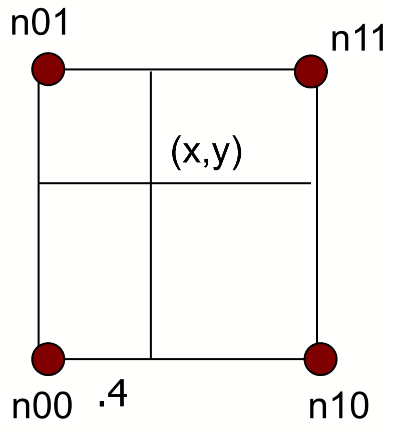
- A noise initialization to set up the noise lattice
  - Create a fixed array of n pseudorandom noise values: **noiseTable**
    - Array of 256 or 512 is sufficient
    - Can be created with standard C function rand() scaled to lie in range[0,1]
- latticeNoise(i,j,k) indexes noise table in a repeatable manner
  - Use a hash function to scramble i,j,k input into an index
    - Minimizes any patterns seen in the noise values
  - Can do this with a second array of n values randomly permuted: **indexTable**
    - e.g. for n = 8: {2, 6, 4, 0, 5, 1, 7, 3}
  - Peachey suggests using 2 methods / macros
    - int PERM(int x) { return indexTable[x & 255]; }
      - Retains only the low order 8 bits
    - int INDEX(int ix, int iy, int iz) { return PERM(ix + PERM(iy+ PERM(iz));}

```
float latticeNoise(int i, int j, int k) { return noiseTable[INDEX(i, j, k)]; }
```

Linear Interpolation of Lattice Noise

- Implement a function noise(x,y,z) that produces a pseudorandom value for any x,y,z
  - Want noise to vary smoothly as x,y,z vary
- Linear interpolation between lattice values produces acceptable results
  - Perlin and Peachey suggest that cubic interpolation provides more realistic results
  - lerp(f, A, B) { return (A + f * (B- A)); }
- Sample noise method follows (from FS Hill)
  - Input a point and a scale value
    - Scale value and offset to ensure all components are positive
      - Does not impact the nature of the noise generated
  - Generate noise values at the eight lattice positions that surround the vertex
    - Perform linear interpolation

2D Example




```
Interpolate in x along y=0 and y=1
n(0.4,0)=lerp(0.4, n00, n10)
n(0.4,1)=lerp(0.4, n00, n10)
Then interpolate  these in y
n(0.4,0.6)=lerp(0.6,n(0.4,0), n(0.4,1)
```

Sample Noise Method

```
float noise(const Point3& p, const float scale)  {
     Point3 pp(p.x*scale+10000, p.y*scale+10000, p.z*scale+10000);
     int ix   = (int)pp.x;          // Integer x
     float tx = pp.x - ix;          // Fractional part
     int iy = (int)pp.y;            // Integer y
     float ty = pp.y - iy;          // Fractional part
     int iz = (int)pp.z;            // Integer z

     // Get noise at 8 lattice points
     float d[2][2][2];
     for (int k = 0; k < 2; k++)
        for (int j = 0; j < 2; j++)
           for (int i = 0; i < 2; i+)
              d[k][j][i] = latticeNoise(ix+i, iy+j, iz + k);

     // Linear interpolation
     float x0 = lerp(tx, d[0][0][0], d[0][0][1]);
     float x1 = lerp(tx, d[0][1][0], d[0][1][1]);
     float x2 = lerp(tx, d[1][0][0], d[1][0][1]);
     float x3 = lerp(tx, d[1][1][0], d[1][1][1]);
     float y0 = lerp(ty, x0, x1);
     float y1 = lerp(yy, x2, x3);
     return lerp(pp.z - iz, y0, y1);
  }
```

Scaled Noise Functions

- Many noise sampling functions require scaling the noise parameters
  - Can also add an offset
  - Scaled noise

$$scaledNoise(s, x, y, z) = noise\big((s * x), (s * y), (s * z)\big)$$

  - Scaled and Offset noise

$$scaledOffsetNoise(s, o, x, y, z) = noise\big((s * x + o), (s * y + o), (s * z + o)\big)$$

Turbulence

- Common use of noise is with Perlin's turbulence function
  - Perlin used it to provide the appearance of marble
    - Perturbed sine wave to modulate color
  - Turbulence adds instances of Perlin noise
    - Creates a fractal character
- Turbulence mixes together several noise components
  - One that fluctuates slowly, one that fluctuates twice as fast, one that fluctuates four times as fast, etc.
    - Each successive term is given a smaller strength
    - For example:

$$turbulence(s,x,y,z) = \frac{1}{2}noise(s,x,y,z) + \frac{1}{4}noise(2s,x,y,z) + \frac{1}{8}noise(4s,x,y,z)$$

    - Parameter s scales distances

Turbulence

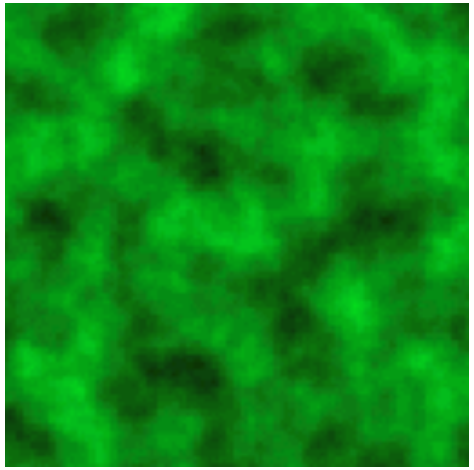
- Perlin's turbulence function takes a position and returns a turbulent scalar value
  - Written in terms of a progression:

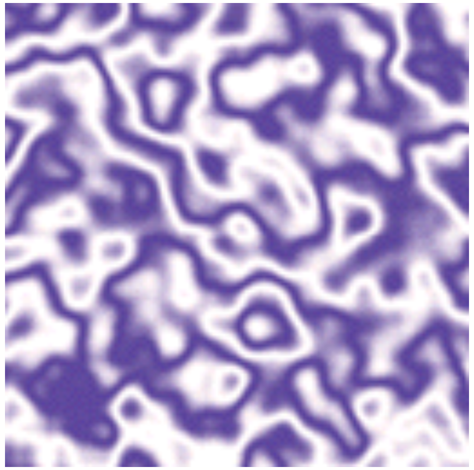
$$turbulence(s, x, y, z) = \sum_{k=0}^{M} \frac{1}{2^k} noise(2^k, s, x, y, z)$$

  - M is the smallest integer satisfying $1 / (2^{M+1}) < $ size of a pixel
    - Truncation limits the function to ensure proper anti-aliasing
- Between successive terms the noise function will vary twice as fast in the second as the first
  - Will contain features that are half the size
  - At each detail scale the amount of noise added to the series is proportional to the detail scale and inversely proportional to the frequency of the noise
    - Self similarity

Turbulence

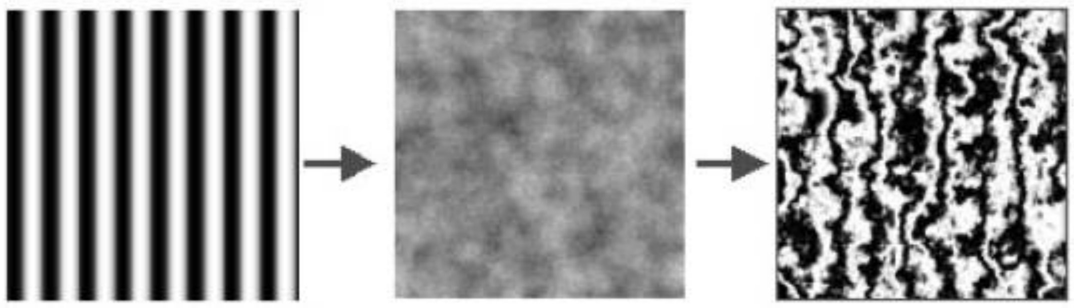
- Simulating turbulence is a two stage process
  - Represent the basic, first-order structural features of the texture through some functional form
    - Function is typically continuous and contains significant variations in its first derivative
      - Example: sin
    - Addition of second and higher order detail by using turbulence to perturb the parameters of the function





Using Turbulence to Generate Marble Textures

- Perlin described turbulation of a sine wave to give the appearance of marble
  - Unperturbed color veins described by a sine wave and an intensity map
  - marble(x)=MarbleColor(sin(x))
    - MarbleColor is a spline curve - mapping a scalar input to intensity
  - Add turbulence
    - marble(x)=MarbleColor(sin(x+turbulence(x)))



Flame

- Watt and Watt describe using turbulence to create animated flame
  - Using a turbulence function defined over time
- Define a general flame shape: minimax coordinates (-b,0), (b,h) in x,y plane
  - Define flame color: 3 spline curves mapping R,G,B intensity
    - Maximum intensity at x = 0 (flame center): falling off to 0 at x = 1
    - Blue and green fall off faster than red
    - Weight color based on height from the base to get variation in y
  - Apply turbulence
    - flame(x,t) = (1 - y/h) flame_color(abs(x/b) + turbulence(x,t))
  - Apply flame() to color a rectangular polygon covering the flame

Image Texture Mapping and Ray Tracing

- Need to associate an inverse mapping from object coordinates (x,y,z) to texture coordinates
  - Requires different mappings for different shapes
- Example: square
  - Simple scaling and translation
- Example: sphere
  - Convert x,y,z to latitude, longitude
  - Scale lat, lon values to lie in [0,1] texture coordinate range
- Inverse mapping for triangle mesh objects
  - If object is modeled using a triangle mesh AND texture coordinates are associated with each vertex we can use barycentric coordinates to interpolate texture values
    - u,v from the ray/Triangle intersect method
    - Provides weightings to apply to each vertex's texture coordinate