generate “image” on the fly, instead of turbulence: multiple feature sizes

- define texture pattern over 3D domain - 3D space containing the object
- texture function can be digitized or procedural
- for each point on object compute texture from point location in space
- 3D function \( p(x,y,z) \)

Review: Perlin Noise

- coherency: smooth not abrupt changes
- turbulence: multiple feature sizes

```plaintext
function marble(point)
    x = point.x + turbulence(point);
    return marble_color(sin(x))
```


- generate “image” on the fly, instead of loading from disk
- often saves space
- allows arbitrary level of detail

Procedural Textures

Review: Volumetric Texture

- define texture pattern over 3D domain - 3D space containing the object
- texture function can be digitized or procedural
- for each point on object compute texture from point location in space
- 3D function \( p(x,y,z) \)
Review: Generating Coherent Noise
- just three main ideas
- nice interpolation
- use vector offsets to make grid irregular
- optimization
- sneaky use of 1D arrays instead of 2D/3D one

Review: Procedural Modeling
- textures, geometry
  - nonprocedural: explicitly stored in memory
- procedural approach
  - compute something on the fly
  - not load from disk
  - often less memory cost
  - visual richness
  - adaptable precision
- noise, fractals, particle systems

Review: Fractal Terrain
- 1D: midpoint displacement
  - divide in half, randomly displace
  - scale variance by half
- 2D: diamond-square
  - generate new value at midpoint
  - average corner values + random displacement
  - scale variance by half each time

Review: Language-Based Generation
- L-Systems
  - F: forward, R: right, L: left
  - Koch snowflake:
    \( F = FLFRFRFLF \)
  - Mariano’s Bush:
    \( F=FF[-F+F+F]_s[-F-F-F] \)
  - angle 16

Fractal Terrain
- also can make using Perlin noise
  - http://mrlnyu.edu/~perlin/planet/

Procedural Approaches

http://spanky.triumf.ca/www/fractint/sys/plants.html

http://www.gameprogrammer.com/fractal.html

http://mrl.nyu.edu/~perlin/planet/
Particle Systems

- loosely defined
  - modeling, or rendering, or animation
- key criteria
  - collection of particles
  - random element controls attributes
    - position, velocity (speed and direction), color, lifetime, age, shape, size, transparency
    - predefined stochastic limits: bounds, variance, type of distribution

Particle System Examples

- objects changing fluidly over time
  - fire, steam, smoke, water
- objects fluid in form
  - grass, hair, dust
- physical processes
  - waterfalls, fireworks, explosions
- group dynamics: behavioral
  - birds/bats flock, fish school, human crowd, dinosaur/elephant stampede

Particle Systems Demos

- general particle systems
  - http://www.wondertouch.com
- boids: bird-like objects
  - http://www.red3d.com/cwr/boids/

Particle Life Cycle

- generation
  - randomly within “fuzzy” location
  - initial attribute values: random or fixed
- dynamics
  - attributes of each particle may vary over time
    - color darker as particle cools off after explosion
  - can also depend on other attributes
    - position: previous particle position + velocity + time
- death
  - age and lifetime for each particle (in frames)
  - or if out of bounds, too dark to see, etc

Particle System Rendering

- expensive to render thousands of particles
- simplify: avoid hidden surface calculations
  - each particle has small graphical primitive (blob)
  - pixel color: sum of all particles mapping to it
- some effects easy
  - temporal anti-aliasing (motion blur)
    - normally expensive: supersampling over time
    - position, velocity known for each particle
      - just render as streak
Procedural Approaches Summary

- Perlin noise
- fractals
- L-systems
- particle systems

- not at all a complete list!
  - big subject: entire classes on this alone

Advanced Rendering

Reading

- FCG Chapter 9: Ray Tracing
  - only 9.1-9.7

Errata

- p 155
  - line 1: \( p(t) = \alpha + \beta t \), not \( p(t) = \alpha + \beta t \)
  - equation 5: 2nd term 2\( d'(\alpha - \alpha) \), not 2\( d'(\alpha - \beta) \)

- p 157
  - matrices: \( c_x \rightarrow X, c_y \rightarrow Y, c_z \rightarrow Z \)

- p 162
  - \( r = d - 2(d.n)n \), not \( r = d + 2(d.n)n \)

- p 163
  - eqn 4 last term: \( n \cos \theta \) not \( n \cos \theta' \)
  - eqn 5: no \( \theta \) term at end

Global Illumination Models

- simple shading methods simulate local illumination models
- no object-object interaction
- global illumination models
  - more realism, more computation
- approaches
  - ray tracing
  - subsurface scattering
  - radiosity

Simple Ray Tracing

- view dependent method
- cast a ray from viewer’s eye through each pixel
- compute intersection of ray with first object in scene
- cast ray from intersection point on object to light sources
Recursive Ray Tracing

- Ray tracing can handle
  - Reflection (chrome)
  - Refraction (glass)
  - Shadows
- Spawn secondary rays
  - Reflection, refraction
    - If another object is hit, recurse to find its color
  - Shadow
    - Cast ray from intersection point to light source, check if intersects another object

Reflection

- Mirror effects
- Perfect specular reflection

Refraction

- Happens at interface between transparent object and surrounding medium
  - E.g. glass/air boundary
- Snell’s Law
  - \( c_1 \sin \theta_1 = c_2 \sin \theta_2 \)
  - Light ray bends based on refractive indices \( c_1, c_2 \)

Total Internal Reflection

As the angle of incidence increases from 0 to greater angles...

- The refracted ray becomes dimmer (there is less refraction)
- The reflected ray becomes brighter (there is more reflection)
- The angle of refraction approaches 90 degrees until finally a reflected ray can no longer be seen.

http://www.physicsclassroom.com/Class/refrn/U14L3b.html

Ray Tracing Algorithm

```
RayTrace(r, scene)
obj := FirstIntersection(r, scene)
if (no obj) return BackgroundColor;
else begin
  if (Reflect(obj)) then
    reflect_color := RayTrace(ReflectRay(r, obj));
  else reflect_color := Black;
  if (Transparent(obj)) then
    refract_color := RayTrace(RefractRay(r, obj));
  else refract_color := Black;
  return Shade(reflect_color, refract_color, obj);
end;
```

Basic Ray Tracing Algorithm

```
RayTrace(r, scene)
if (no obj) return BackgroundColor;
else begin
  if (Reflect(obj)) then
    reflect_color := RayTrace(ReflectRay(r, obj));
  else reflect_color := Black;
  if (Transparent(obj)) then
    refract_color := RayTrace(RefractRay(r, obj));
  else refract_color := Black;
  return Shade(reflect_color, refract_color, obj);
end;
```
Algorithm Termination Criteria
- termination criteria
  - no intersection
  - reach maximal depth
  - number of bounces
  - contribution of secondary ray attenuated below threshold
  - each reflection/refraction attenuates ray

Ray - Object Intersections
- inner loop of ray-tracing
  - must be extremely efficient
- solve a set of equations
  - ray-sphere
  - ray-triangle
  - ray-polygon

Ray - Sphere Intersection
- ray: \( x(t) = p_x + v_x t, \ y(t) = p_y + v_y t, \ z(t) = p_z + v_z t \)
- unit sphere: \( x^2 + y^2 + z^2 = 1 \)
- quadratic equation in \( t \):
  \[
  0 = (p_x + v_x t)^2 + (p_y + v_y t)^2 + (p_z + v_z t)^2 - 1
  = t^2(v_x^2 + v_y^2 + v_z^2) + 2(p_x v_x + p_y v_y + p_z v_z)
  + (p_x^2 + p_y^2 + p_z^2) - 1
  \]

Optimized Ray-Tracing
- basic algorithm simple but very expensive
- optimize by reducing:
  - number of rays traced
  - number of ray-object intersection calculations
- methods
  - bounding volumes: boxes, spheres
  - spatial subdivision
    - uniform
    - BSP trees
  - (not required reading)

Subsurface Scattering: Translucency
- light enters and leaves at different locations on the surface
- bounces around inside
- technical Academy Award, 2003
  - Jensen, Marschner, Hanrahan

Subsurface Scattering: Marble
Subsurface Scattering: Milk vs. Paint

Subsurface Scattering: Faces

Subsurface Scattering: Faces