game|tech 2004

Data Amplification

Jason Mitchell
3D Application Research Group Lead, ATI Research

Overview

- Motivation
- Data Amplification
- Textures
  - Compositing signals and noise, clouds
  - Hybrid textures
  - Wang Tiles
  - Flow-based synthesis
- Geometry Synthesis
  - Ocean Water
  - Particle systems
- Instancing
  - Drawing Crowds
- Other areas
  - Plants
  - Procedural cities
  - Higher order surfaces
- Existence proofs
  - *Grafan* (others in development)

Motivation

- Market demands larger, more complex game worlds
- GPUs can consume ever increasing amounts of data, producing high fidelity images

- Two major issues
  1. Cost of authoring these datasets is increasing
  2. Mass storage (Hard Drive and DVD) and volatile storage (RAM) are slow and small compared to the ability of the GPU to consume data

- *Amplify* the data
  - Get the most out of what we build
  - Get the most out of what we have loaded in memory at any given time

[Smith84] – Plants, Fractals and Formal Languages

- SIGGRAPH paper which coined the term *database amplification*
- Discussion of plant and mountain growth using L-systems and particle systems
- Two exciting properties:
  1. **Database amplification** – Complex images from small datasets
     - If you can generate it, an artist doesn't have to build it
     - Consoles and PCs have limited memory relative to processing power
     - Network bandwidth is limited. Would be nice to "grow" data from seeds sent across the wire. Has LOD opportunities built right in.
  2. **Emergence** – Complex appearance from simple rules
     - Can generate more variety and volume than an artist could ever build

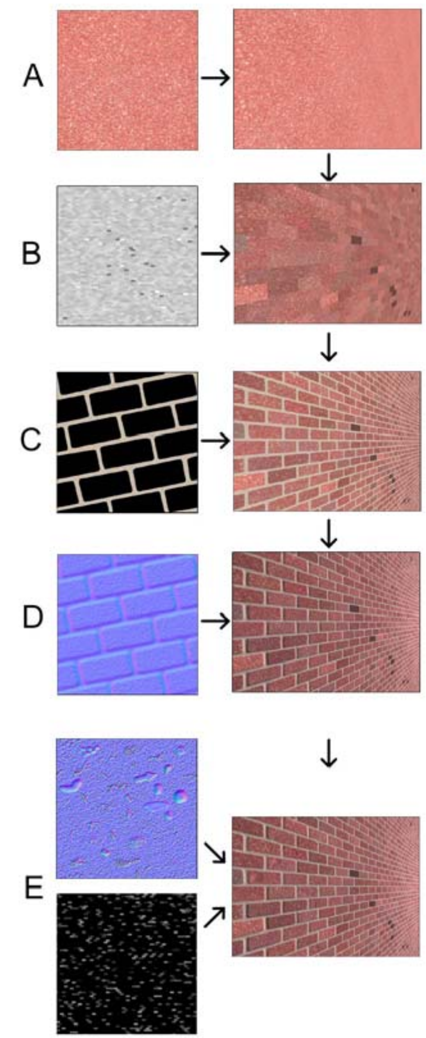
Procedural Textures

- Combine signals at different frequencies to make more stuff
- Examples
  - Clouds
  - Hybrid procedural and authored approaches
  - Wang tiles
  - Flow-bases synthesis
  - Fourier-domain water synthesis

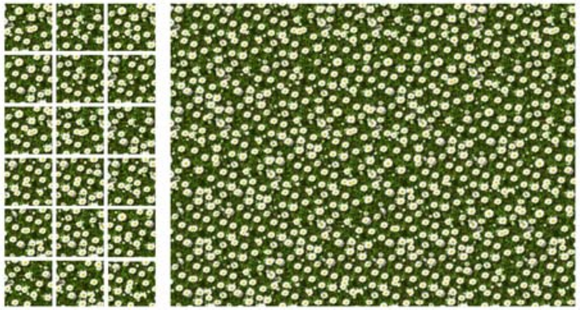
From [Dube05]

Hybrid procedural approaches

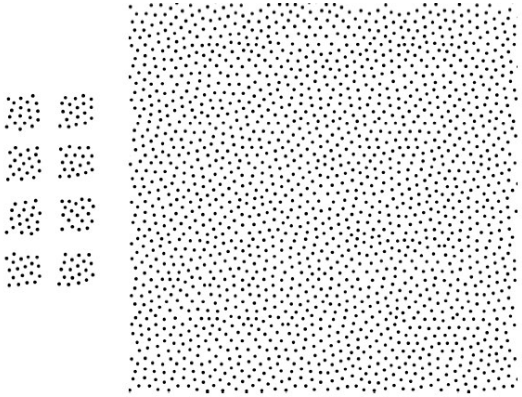
- In a recent *Game Developer Magazine* article, Sean Barrett discusses *Hybrid Procedural Textures*
  - Find a middle ground between sampling and synthesis
- You may already be doing something like this with *detail textures*



[Cohen03] Wang Tiles



- Set of square texture maps which can be used to tile a surface



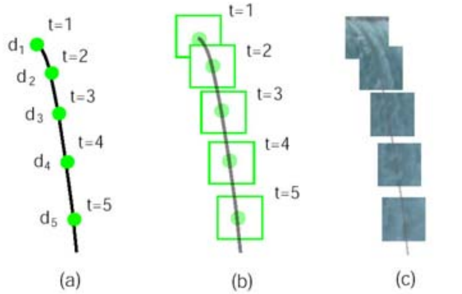
[Bhat04] Flow-based Video Synthesis and Editing

> Analyze real-world video

> Use a particle model to synthesize video/texture of continuous flow

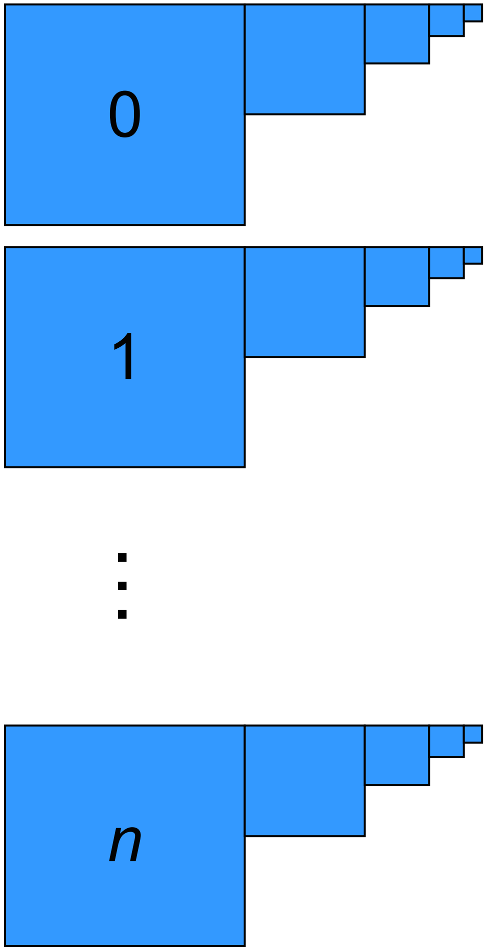
> Could also think of this as a kind of compression

> Could integrate naturally into many 3D scenes



Texture Arrays

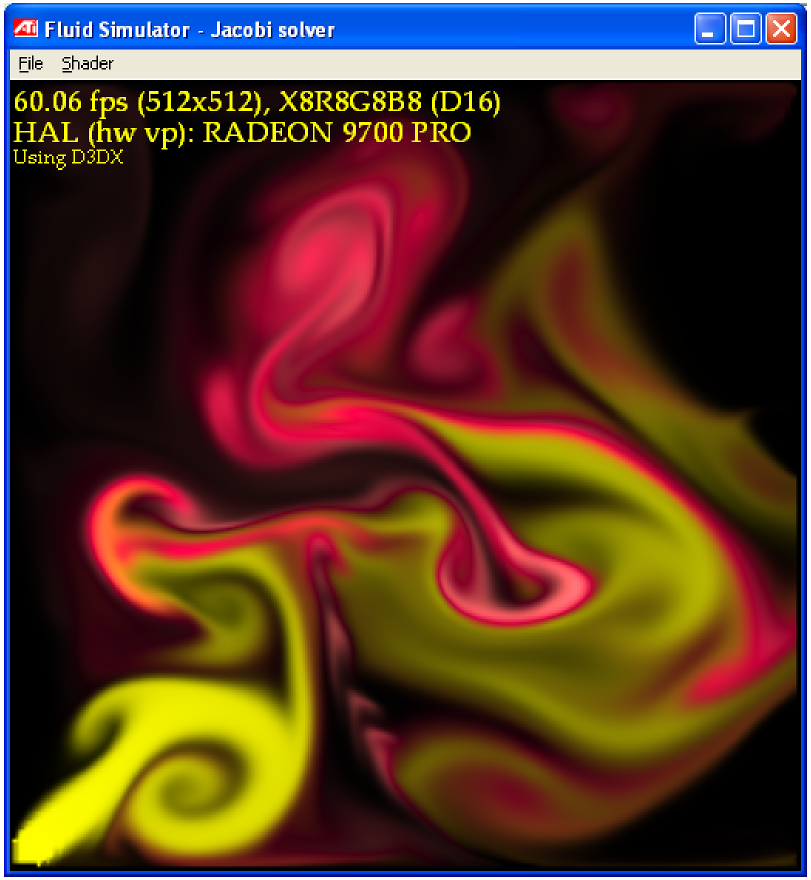
- New type of texture construct coming in the future
- These are *not* volume textures
  - Mip-mapping is different
- Pixel shader sampling instruction specifies texture coordinates and array index in argument



Fluid Navier-Stokes equations

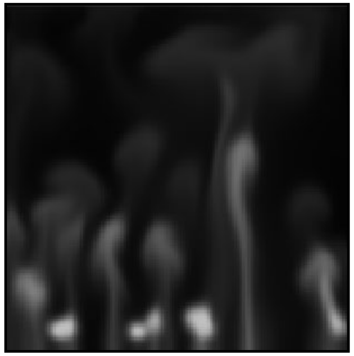
> It is now possible to do 2D fluid simulations on GPUs

> Can be useful for generating decorative smoke wisps



Integration into scene

- Obviously, this doesn't have to be physically accurate, just plausible
- Once you have the implementation and the GPU cycles to burn, you can drop this sort of thing in anywhere



Geometry Amplification

- It’s easy to play games with textures using pixel shaders, but how do we amplify our geometry?
- Synthesis
  - Make more!
- Instancing
  - Reuse the data in interesting ways which hide the replication

Geometry synthesis

- Textures are easy to generate using pixel shaders as image processing kernels, but we want to process geometry too
- For certain 1:1 or many:1 operations, GPU-based geometry processing and generation is real
  - Really it has been around a while, but the APIs are in the way
- Want to synthesize data on demand rather than store a lot of it
  - This includes geometry!

On-demand Synthesis of Water

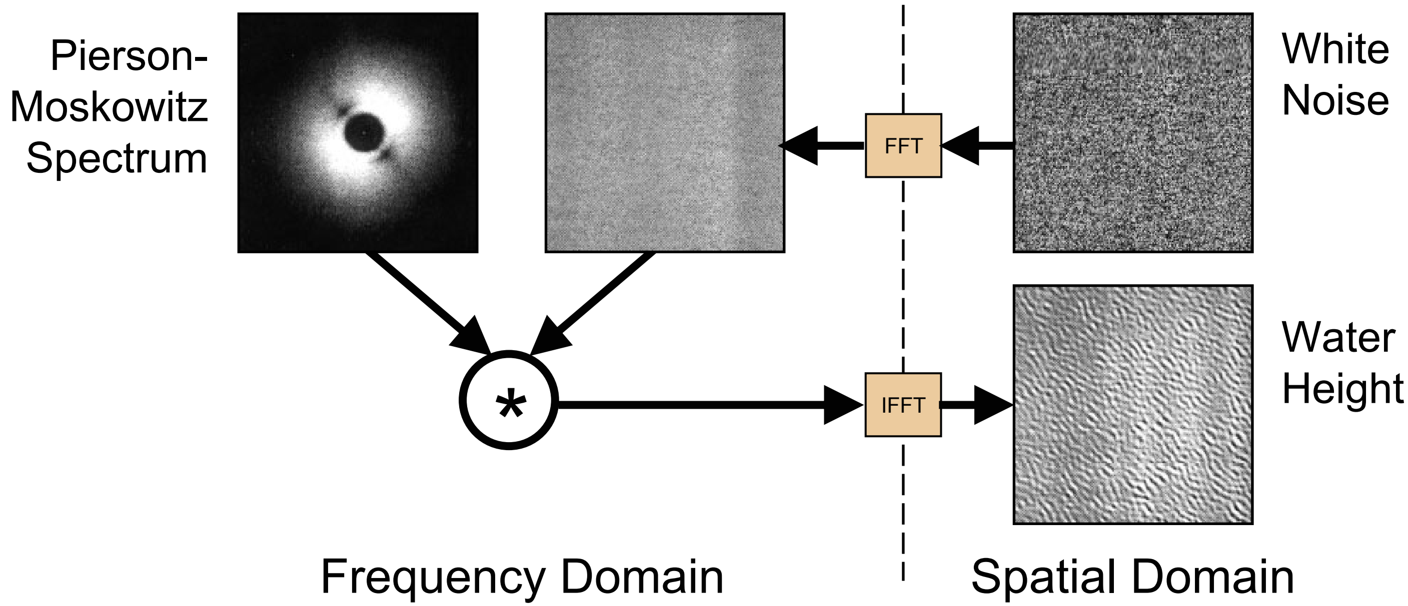
- Storing lots of precomputed water animation takes up lots of memory
  - Would be nice if it could be generated on demand
- Computing water animation via realistic simulation in real-time is expensive
  - It just has to be plausible
- Simply scrolling noise can look OK, but we want to do better
  - We’ve done scrolling noise in the past, but we can do better

Two Classes of Approach

- Spatial domain
  - Compute superposition of a finite set of waveforms directly
  - Can be sinusoids or trochoids or something more arbitrary
- Fourier domain
  - Synthesize and animate spectrum of ocean water
  - Take IFFT to get height and normal maps

[Mastin87] Fourier Synthesis of Ocean Scenes

- Transformed white noise to the Fourier domain and then filtered it using a spectrum which resembles ocean water
  - Used the Pierson-Moskowitz spectrum which was derived from real ocean wave measurements
  - Relates wind speed to spectrum of sea
- Inverse FFT of the filtered result produces a tileable height map which resembles ocean waves
- Can portray wave motion by manipulating the phase

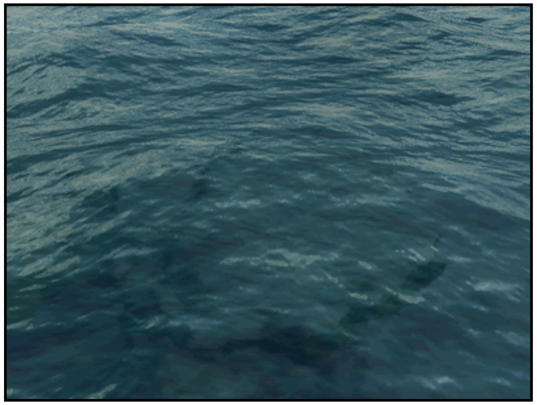


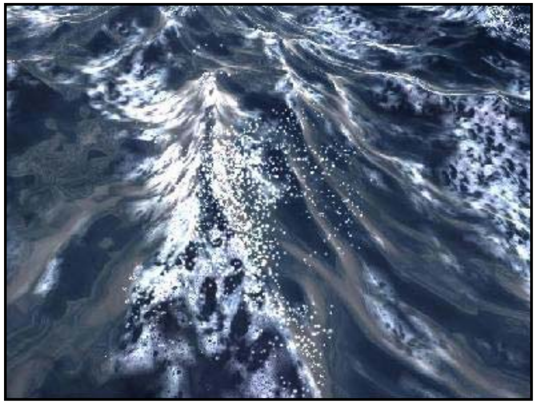
[Tessendorf99] Simulating Ocean Water

- Did water for *Waterworld*, *Titanic* and many others
- Works with sums of sinusoids but starts in Fourier domain
- Can evaluate at any time *t* without having to evaluate other times
- Uses the Phillips Spectrum and describes how to tune it to get desired looks
  - Roughness of the sea as a function of wind speed
  - Directional dependence to simulate waves approaching shore

[Jensen01] Deep-Water Animation and Rendering

- Adopted many techniques from Tessendorf, all in real time
- Used low frequencies to displace geometry and high frequencies in a normal map
- First attempt at Fourier synthesis of ocean water in real time, but IFFT was done on the CPU
- Also played with all sorts of other things like foam, spray, caustics and godrays





FFT on the GPU

- A couple of different GPU-based FFT implementations have been developed in the last few years
  - Some colleagues and I published an implementation of Cooley and Tukey's "Decimation in Time" algorithm, which we published in an image processing chapter in ShaderX^2 [Cooley65] [Mitchell03] .
  - [Moreland03] also published a paper on doing the FFT on a GPU

Migrate it all to the GPU

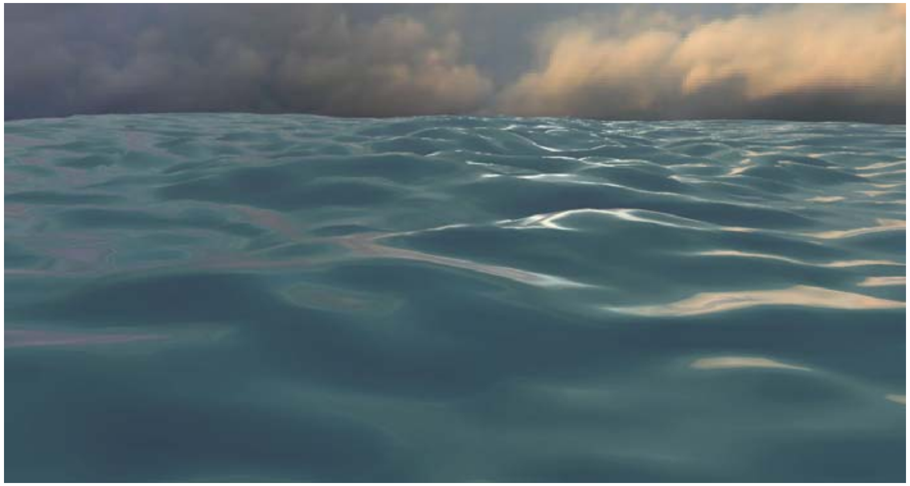
> If we can now do an FFT on the GPU, let’s do everything on the GPU

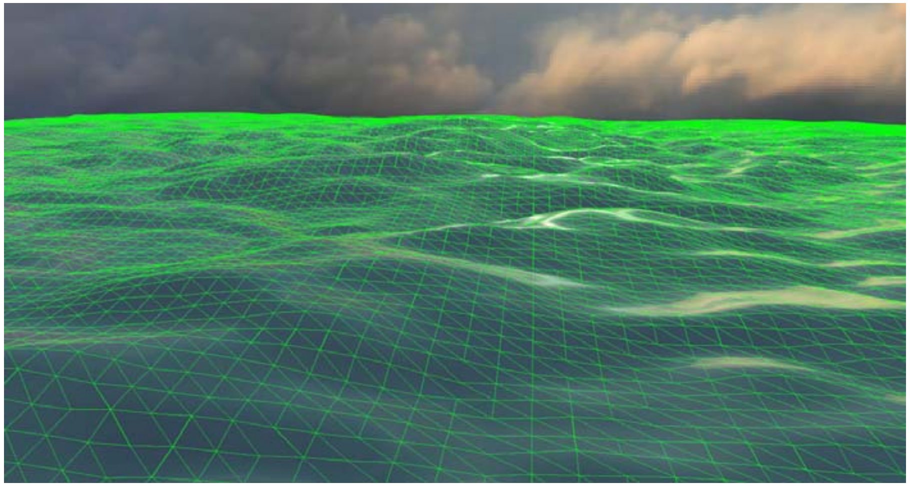
> The algorithm:

1. Load initial frequency data to static textures
2. For each frame
   a. Generate Fourier spectrum at time *t*
   b. Do IFFT to transform to spatial domain height field
   c. Filter height field to generate normal map
   d. Cast height field to vertex buffer to use as displacement stream
   e. Render mesh tiles using displacement stream and normal map to shade

Synthesized Water

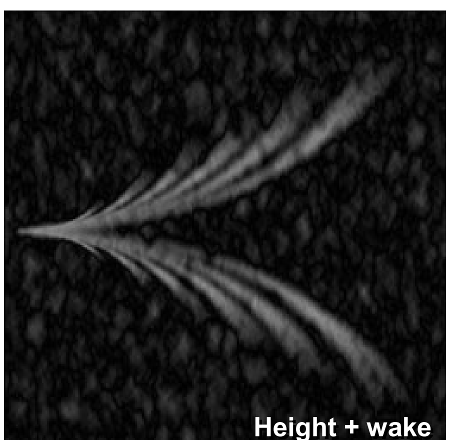
- Apply synthesized height field to vertices and displace vertically
- Filter to create a normal map for shading

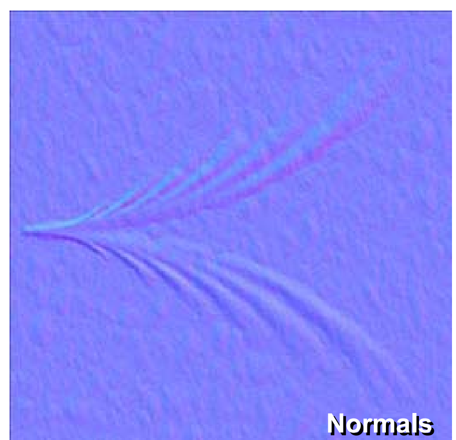




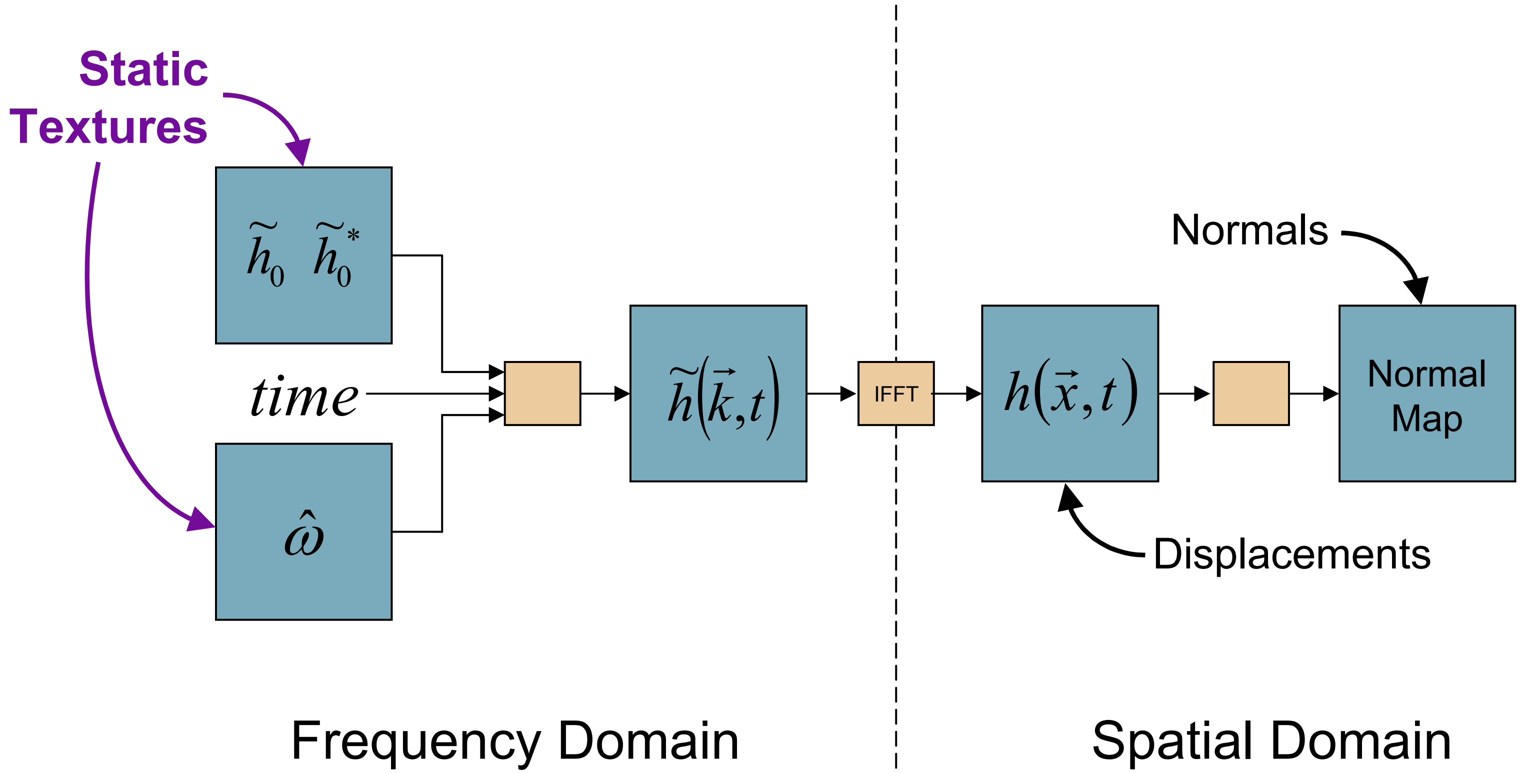
Additional waveforms

- Easy to composite wake, eddies, simulation etc
- Precomputed waveforms or real-time simulation like the Navier-Stokes simulation demonstrated earlier
- Then filter to get normals for shading

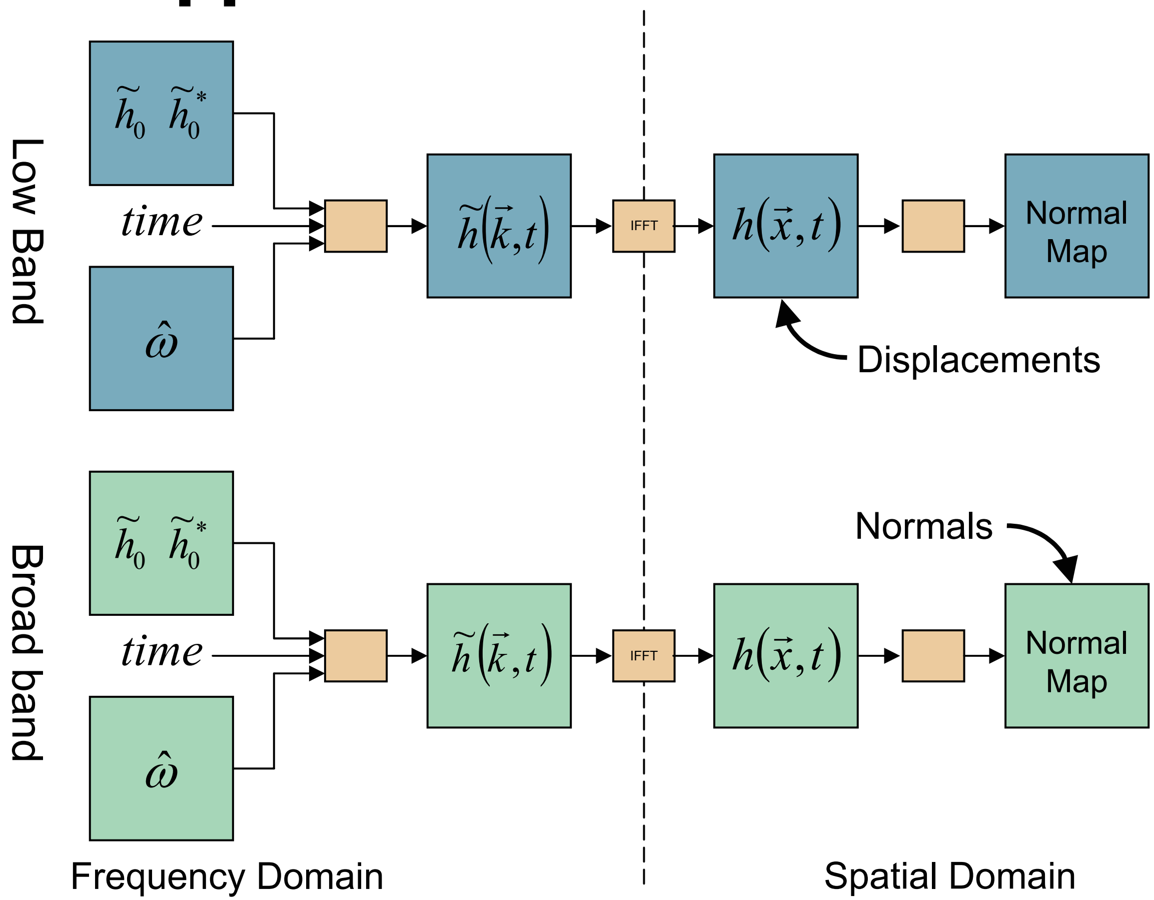




Single-band approach

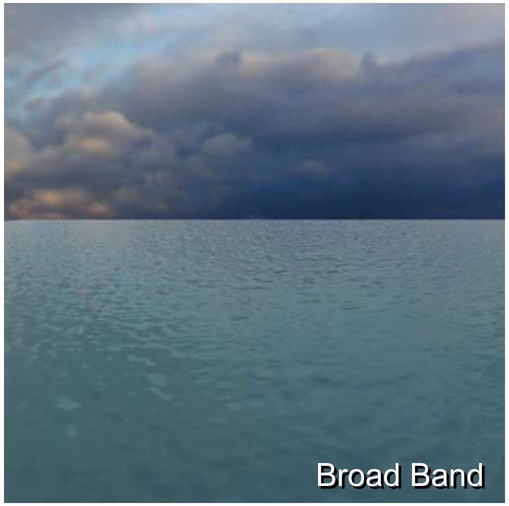


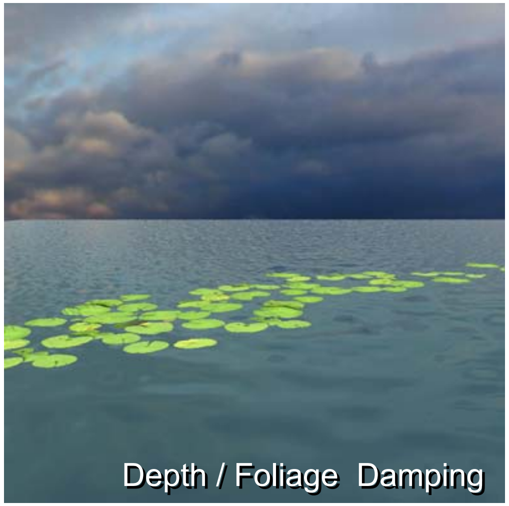
Dual-band approach



Depth effects

- Shallow areas or foliage can damp out high frequencies
- Simply blend between broad and low band maps to approximate the look





Caustics

> Patterns caused by convergence of refracted or reflected light
> Important visual cue in certain scales of water rendering
>> Refracted caustics in swimming pool or other shallow water
>> Reflected caustics on boat hull

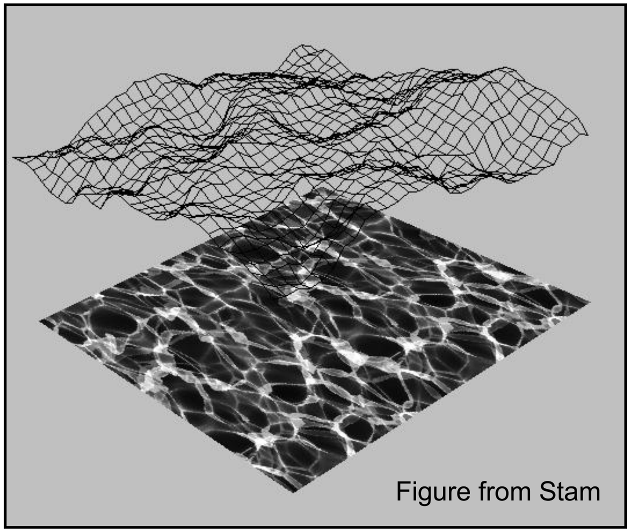


Figure from Stam

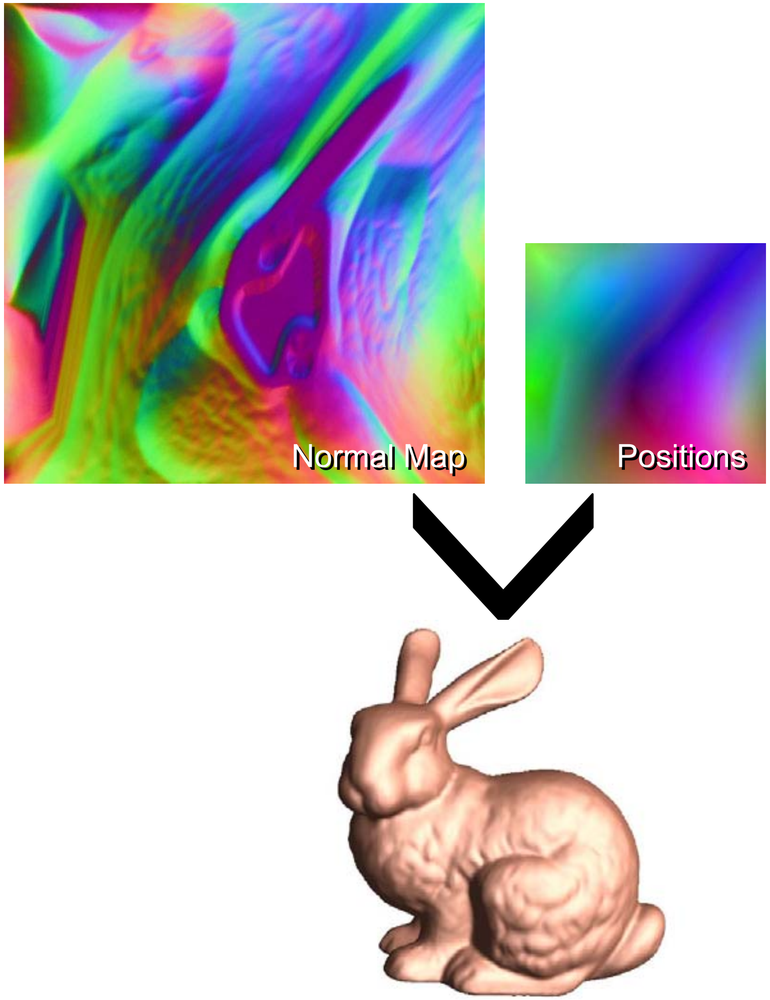
Reflected Caustics

Interaction

- If the GPU does the amplification, what does this do to our interactions with the world, which are simulated on the CPU?
  - Multi-resolution synthesis (low resolution on CPU for gross collision interaction & high resolution on GPU for rendering)

[Gu02] Geometry Images

- Reparametrize mesh into square grid
- Since neighbors are implicit, it's easy to process in this space using image processing concepts
- Reconstruct processed geometric model



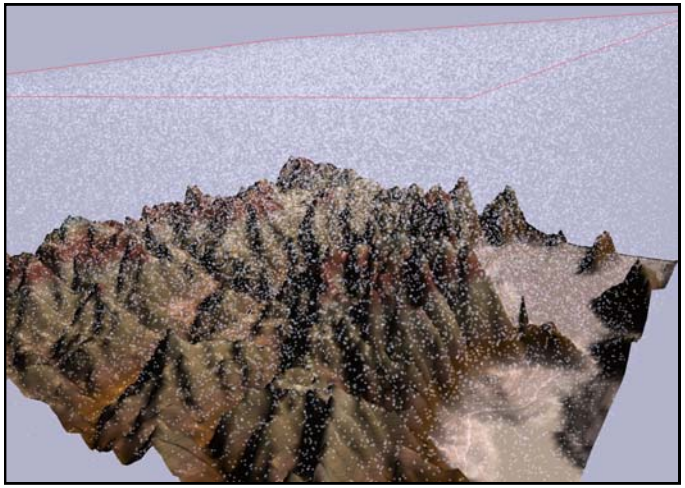
[Reeves85] Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems

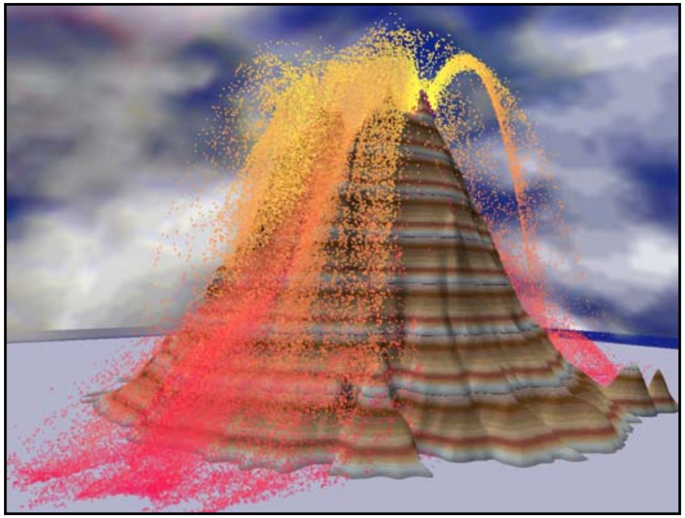
- Topology procedurally generated, with properties tuned to resemble known tree species
- Pixar used a *structured* particle system approach to rendering foliage in *The Adventures of Andre and Wally B*
- Up to **3000x data amplification**
- Polygonal trunk / branches
- One particle system per tree
  - Circles and Lines
- Simple procedural shading, taking into account depth into the tree and gross occlusion by neighboring tree bounds

Particle Systems

- In [Kipfer04], the authors use the GPU to implement a particle engine they call Uberflow
- Particle-Particle and Particle-Scene collisions
- Can sort back to front
- Measure the following perf in frames per second:

	No collisions	Collisions with height field	Particle-Particle collisions	Sorting, but no collisions	CPU sorting, no collisions
256^2	640	155	133	39	7
512^2	320	96	31	8	2
1024^2	120	42	7	1.4	0.4





Instancing and Variation

- Want to use a single source model at multiple physical locations in an environment
- The best way to handle groups of similar things
  - Foliage, crowds
- Ideally done with one API call and no data replication
  - Direct3D has recently added an instancing capability
- Use shaders to generate uniqueness across instances with spatially varying parameters

Instancing in Practice

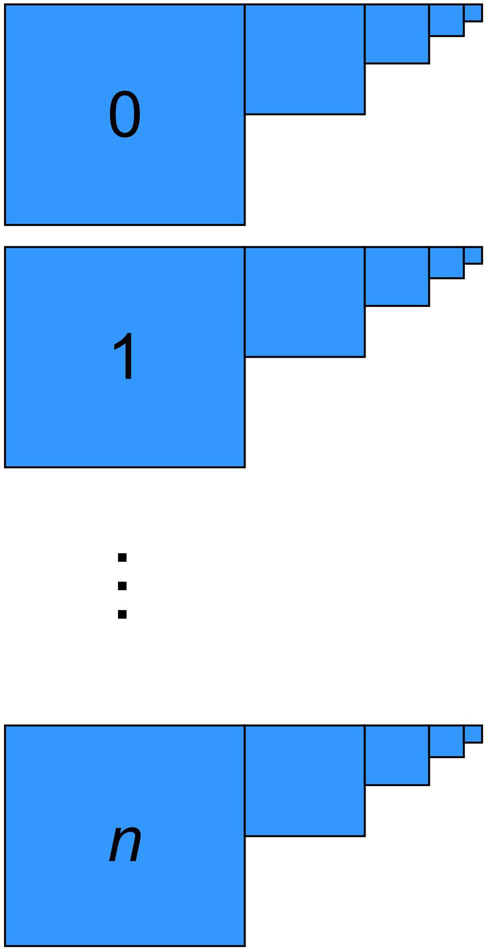
- New API in Direct3D
  - Store per-instance data in a separate data stream
  - Draw multiple instances in one shot
- Example from *Far Cry*, for a representative forest scene:
  - Approximately 24 kinds of vegetation
  - 4 types of grass (45 to 120 polygons per instance)
  - 12 types of bushes (100 to 224 polygons per instance)
  - 8 types of trees (500 to 1600 polygons per instance)
  - Instancing on some scenes is very efficient. Number of **draw-calls** (including other non-instanced objects) is **reduced from 2500 to 430**
  - Far Cry doesn't seem to do much sorting of foliage, which is one reason this works

Hundreds of instanced characters

Unique seeds for instanced shading

Texture Arrays for Instancing

> Useful for providing unique look to instanced objects
  > Index into an array of textures based upon some “state” stored with each instance (like color seeds on previous slide)
  > Same “state” can be used to drive flow control as well
  > Like being able to change texture handles mid draw call



Procedural Environments

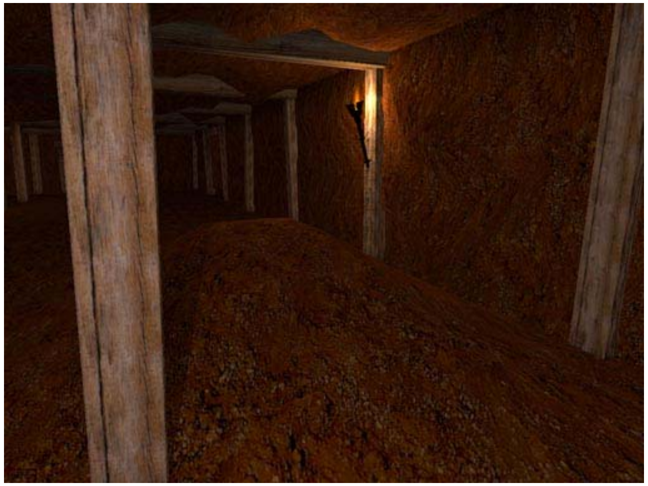
- 2D games have done this for years
- The demoscene does a ton of this
- Some games are now extending to 3D:
  - Author time
  - Run time
    - *Grafan*

Demoscene

- To fit into 64kb or 96kb, the demoscene guys are doing a lot of procedural generation
- The 96kb game winner at *Breakpoint 2004* (.kkrieger by .theprodukkt) uses a lot of procedural generation and they have even posted their tools online

Grafan from Emogence

- Environments are procedurally generated on-the-fly as a user traverses the game world
- Objects and entities are placed procedurally as the game world is built
- Assets like textures, objects, and entities are authored by artists
- Computation replaces bandwidth



Related areas

- Higher Order Surfaces
  - This is a massive topic that we don’t have time for here, but…
- Plants
- Terrain

Summary

- Textures
  - Compositing signals and noise, clouds
  - Hybrid textures
  - Wang Tiles
  - Flow-based synthesis
- Geometry Synthesis
  - Ocean Water
  - Particle systems
- Instancing
  - Drawing Crowds
- Other areas
  - Plants
  - Procedural cities
  - Higher order surfaces
- Existence proofs
  - *Grafan* (others in development)

References

- [Barrett04] Sean Barrett, “Hybrid Procedural Textures,” *Game Developer Magazine*, October 2004
- [Bhat04] Kiran S. Bhat, Steven M. Seitz, Jessica K. Hodgins and Pradeep K. Khosla, “Flow-based Video Synthesis and Editing,” SIGGRAPH 2004.
- [Cohen03] Michael F. Cohen, Jonathan Shade, Stefan Hiller and Oliver Deussen, “Wang Tiles for Image and Texture Generation,” SIGGRAPH 2003, July, 2003
- [Cooley65] James W. Cooley and John W. Tukey, “An Algorithm for the Machine Calculation of Complex Fourier Series.” Math. Comput. 19, 297-301, 1965.
- [Dube05] Jean-François Dubé, “Realistic Cloud Rendering on Modern GPUs” in *Game Programming Gems 5*, Charles River Media 2005
- [Ebert03] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and Steven Worley, *Texturing and Modeling: A Procedural Approach*, Morgan Kaufmann 2003.
- [Gosselin04] David Gosselin, Pedro V. Sander and Jason L. Mitchell, "Drawing a Crowd" in ShaderX3, Charles River Media, 2004.
- [Gu02] Geometry Images
- [Jensen01] Lasse Staff Jensen and Robert Goliáš, “Deep-Water Animation and Rendering,” Game Developers Conference Europe, 2001. http://www.gamasutra.com/gdce/2001/jensen/jensen_01.htm
- [Kipfer04] Peter Kipfer, Mark Segal and Rüdiger Westermann , “UberFlow: A GPU-Based Particle Engine,” Graphics Hardware 2004
- [Mastin87] Gary A. Mastin, Peter A. Watterger, and John F. Mareda, “Fourier Synthesis of Ocean Scenes,” *IEEE Computer Graphics and Applications*, March 1987, p. 16-23.
- [Mitchell03] Jason L. Mitchell, Marwan Y. Ansari and Evan Hart, “Advanced Image Processing with DirectX 9 Pixel Shaders” in *ShaderX 2 - Shader Tips and Tricks*, Wolfgang Engel editor, Wordware, Sept. 2003.
- [Moreland03] Kenneth Moreland and Edward Angel, “The FFT on a GPU,” *SIGGRAPH/Eurographics Workshop on Graphics Hardware 2003 Proceedings*, pp. 112–119, July 2003.
- [Perlin85] Ken Perlin, “An Image Synthesizer,” SIGGRAPH 1985.
- [Tessendorf99] Jerry Tessendorf, “Simulating Ocean Water,” *Simulating Nature: Realistic and Interactive Techniques* Course Notes, SIGGRAPH 1999.