
frostbite
Rendering Architecture and
Real-time Procedural Shading & Texturing Techniques
Johan Andersson, Rendering Architect, DICE
Natalya Tatarchuk, Staff Research Engineer,
3D Application Research Group, AMD Graphics Products Group
DICE
GameDevelopers® Conference 07
AMD
ATI RADEON GRAPHICS

Outline

- Introduction
- Frostbite Engine
- Examples from demos
- Conclusions

Outline

- **Introduction**
- Frostbite Engine
- Examples from demos
- Conclusions

Complex Games of Tomorrow Demand High Details and Lots of Attention

- Everyone realizes the need to make immersive environments
- Doing so successfully requires many complex shaders with many artist parameters
- We created ~500 custom unique shaders for ToyShop
- Newer games and demos demand even more
  - Unique materials aren’t going to be a reasonable solution in that setting
  - We also need to enable artists to work closely with the surface materials so that the final game looks better
- Shader permutation management is a serious problem facing all game developers

Why Do We Care About Procedural Generation?

- Recent and upcoming games display giant, rich, complex worlds
- Varied art assets (images and geometry) are difficult and time-consuming to generate
  - Procedural generation allows creation of many such assets with subtle tweaks of parameters
- Memory-limited systems can benefit greatly from procedural texturing
  - Smaller distribution size
  - Lots of variation
  - No memory/bandwidth requirements

Procedural Helps You Avoid the Resolution Problem

- Any stored texture has limited resolution.
  - If you zoom in too closely, you will see a lack of detail
  - Or even signs of the original pixels in the texture
- Procedural patterns can have detail at all scales
  - Zooming in : introduce new high frequency details as you zoom
- Zooming out
  - A prebaked texture will start tiling or show seams
  - A procedural texture can be written to cover arbitrarily large areas without seams or repetition
- No mapping problem
  - Don’t have to worry about texture seams, cracks and other painful parameterization problems
  - Solid textures

Where Did That Tank Go?

- Networked games have to deal with sending assets across the network
  - Sending full content (assets, controls) through the network is not the best plan for interactivity - delays
  - Network bandwidth is not increasing at any close rate to the speed of GPUs and CPUs
- Procedural techniques help with this
  - We can send description in compact form to / from server
    - Master particles
    - Grammar descriptions for objects
    - Etc…
  - Content can be generated directly on the client

Let's Not Forget About Interactivity!

- Real-time rendering is quickly becoming fully realistic
  - Excellent foliage, effects, character rendering
  - Often because we can author suitable static assets
- Interactivity is the next frontier!
  - Game play is the king!
- Games are becoming more and more dynamic
  - They make it look like you can blow up anything anywhere…
- But we can't use static resources and expect the same level of interactivity without price
  - More objects means more draw calls, more memory, more authoring, more textures, more, more, more….
  - Eventually the cost becomes too excessive
- We can generate objects with procedural techniques
  - Then use rules to deform / destroy / modify / move them
  - Better interactivity

Procedural Techniques: Now!

- Computers are fast enough so that procedural is real-time now!
  - Flexible shader models allow us to directly translate many of the offline shaders
- Direct3D10® opened new doors for procedural generation in real-time: *flexibility and power*
  - Convenience of geometry shaders and stream out
  - More flexible use of texture / buffer resources
  - Ability to directly render and filter volume textures
  - Integer and bitwise operations

Outline

- Introduction
- **Frostbite Engine**
- Examples from demos
- Conclusions

Frostbite?

- DICE next-gen engine & framework
- Built from the ground up for
  - Xbox 360
  - PlayStation 3
  - Multi-core PCs
    - DirectX 9 SM3 & Direct3D 10
- To be used in future DICE games

Battlefield: Bad Company

- Frostbite pilot project
- Xbox 360 & PlayStation 3
- Story- & character-driven
- Singleplayer & multiplayer
- Large dynamic non-linear environments
  - = you can blow stuff up ☺

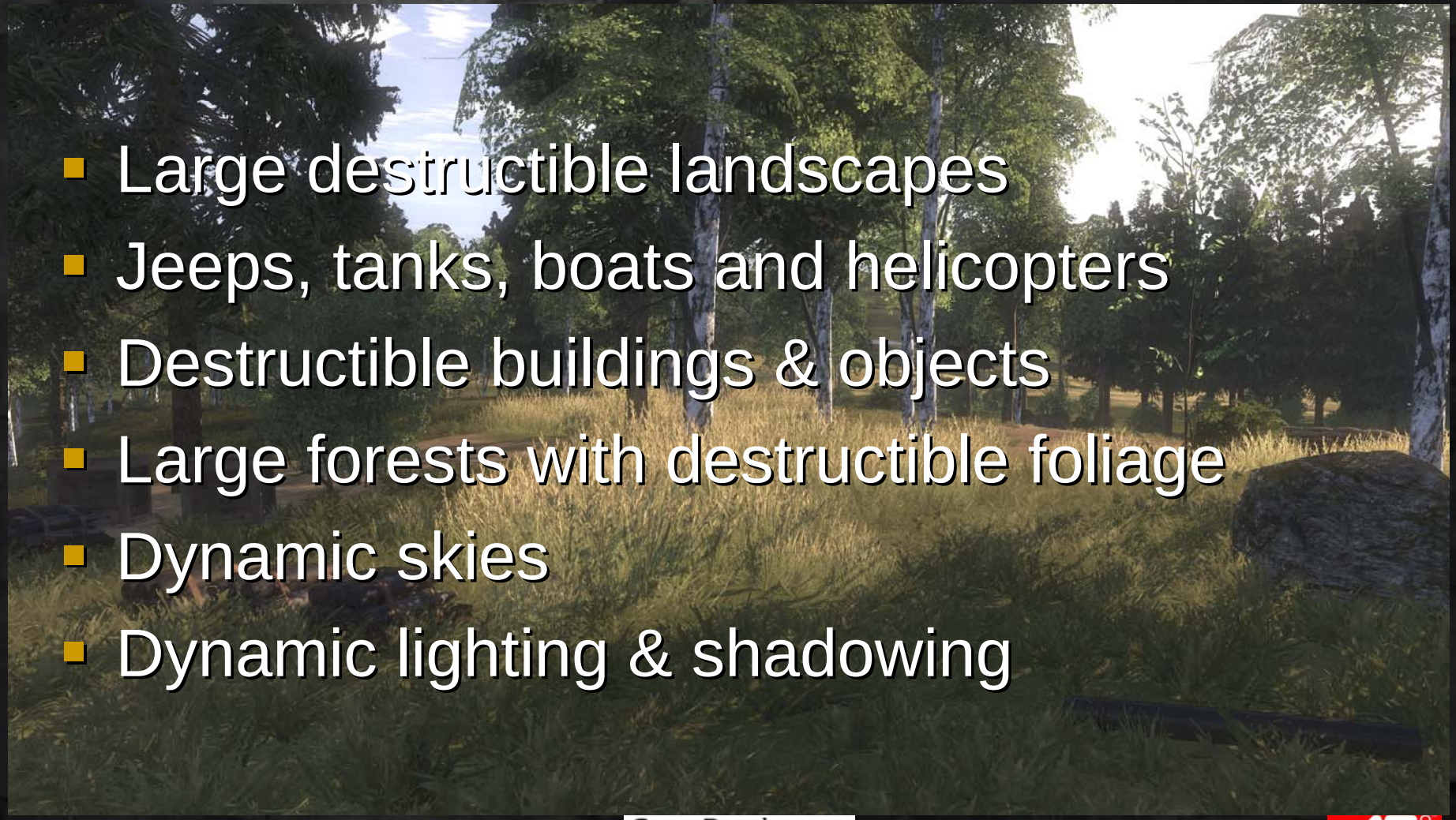
DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Battlefield: Bad Company Teaser

http://media.xbox360.ign.com/media/713/713943/vid_1921226.html

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Battlefield: Bad Company features



- Large destructible landscapes
- Jeeps, tanks, boats and helicopters
- Destructible buildings & objects
- Large forests with destructible foliage
- Dynamic skies
- Dynamic lighting & shadowing

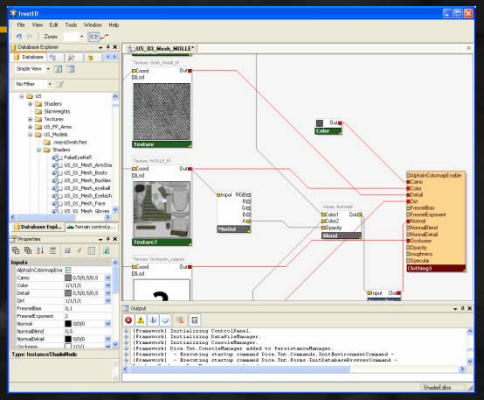
DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

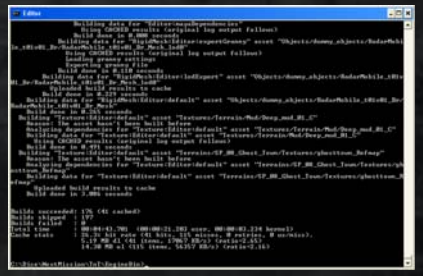
Frostbite design

- Heavily influenced by BFBC features
- Big focus on dynamic memory efficient systems & semi-procedural techniques
  - Due to destruction & non-linear environment
  - But precompute offline whenever possible
- Flexibility and scalability for future needs
  - Not ”only” a Battlefield-engine

Frostbite concepts

- Editor (FrostED)
  - Asset creation
    - Levels, meshes, shaders, objects
  - Fully separate and C#-based
- Pipeline
  - Converts assets to runtime format
  - Win32 only
  - Important for loading times and flexibility
- Runtime
  - ”The Game”
  - Gameplay, simulation, rendering
  - Xbox 360, PS3, Win32





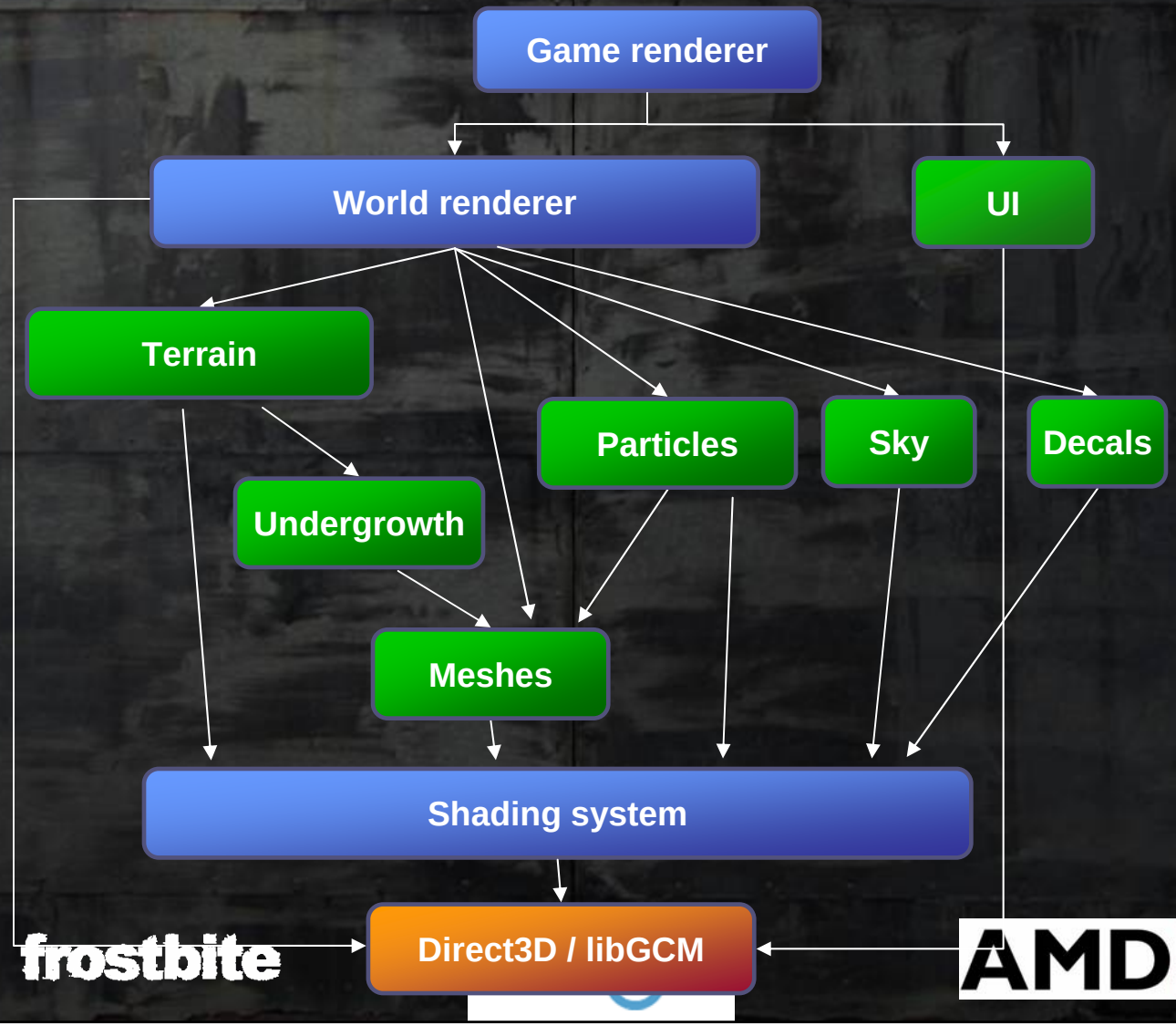
DICE frostbite

GameDevelopers® Conference 07

AMD

ATI Radeon GRAPHICS

Rendering systems overview



DICE frostbite AMD ATI Radeon GRAPHICS

Shading system

- High-level platform-independent rendering API
- Simplifies and generalizes rendering, shading and lighting
  - To make it easy & fast to do high-quality shading
- Handles most of the communication with the GPU and platform APIs

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Shading system backends

- Multiple backends
  - DirectX 9 SM3 for PC & Xbox 360
    - Low-level GPU communication on 360
  - Direct3D 10 for Windows Vista
  - libGCM for PlayStation 3

- Allows other rendering system to focus on what is important instead of platform differences

High-level shading states

- Key feature of shading system
- Rich high-level states instead of low-level platform-dependent states
- More flexible for both user and system

High-level state examples

- Light sources
  - Amount, types, color, shadow
- Geometry processing
  - Skinning
  - Instancing
- Effects
  - Light-scattering, fog
- Surface shaders
  - Instead of vertex & pixel shaders
  - Very powerful

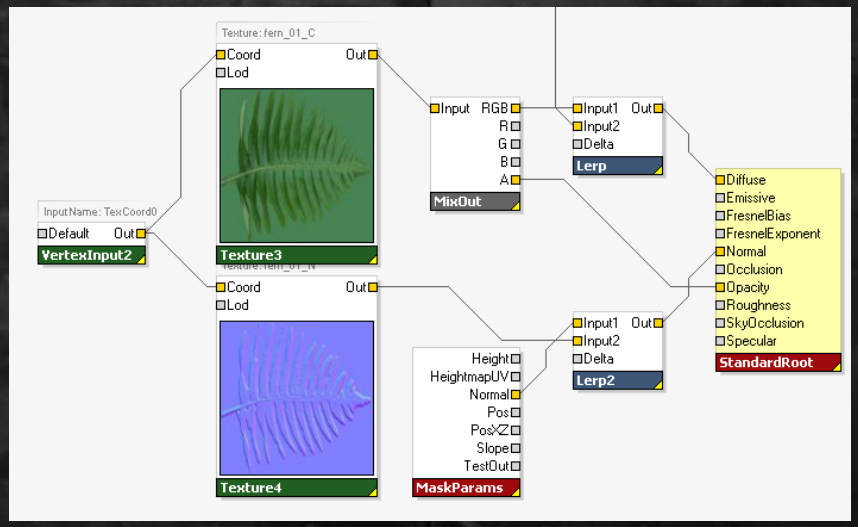
DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

High-level state benefits

- Easier to use and more productive for users
- Share & reuse features between systems
- Hides & manages shader permutation hell
  - Generalized and centralized to shader pipeline
  - Cumbersome manual management in RSC2 & BF2
- Platforms may implement states differently
  - Depending on capabilities
  - Multi-pass lighting instead of single-pass

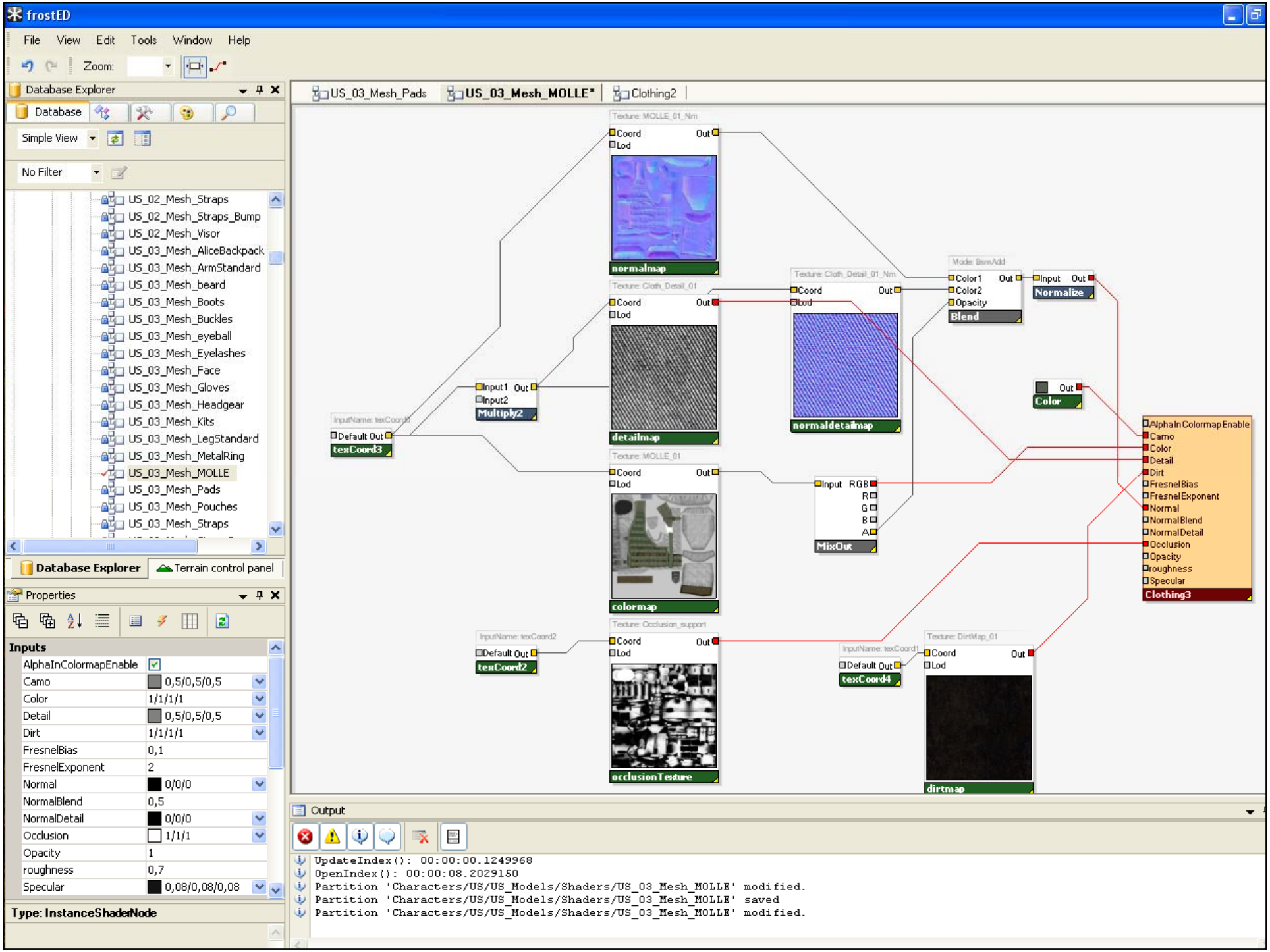
Surface shaders

- Term borrowed from Renderman
- Shader that calculates outgoing color and opacity of a point on a surface
  - Similar to pixel shaders, but not quite..



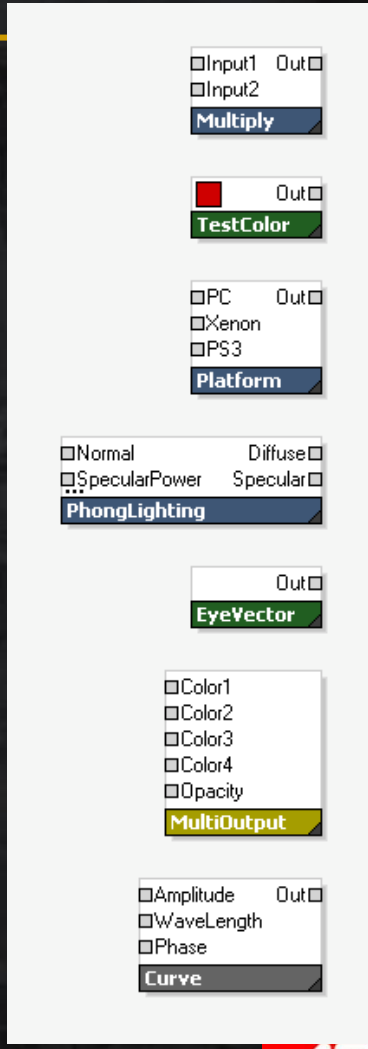
Surface shaders vs pixel shaders

- Graph-based instead of code
  - Easier to build, tweak & manage for artists
- Independent of lighting & environment
- Rich data-centric control flow
  - No need to manually specialize shaders to enable/disable features
- Calculations can be done on any level
  - Per-pixel, per-vertex, per-object, per-frame
  - Split to multiple passes


frostED
File View Edit Tools Window Help
Zoom:
Database Explorer
Database
Simple View
No Filter
US_02_Mesh_Straps
US_02_Mesh_Straps_Bump
US_02_Mesh_Visor
US_03_Mesh_AliceBackpack
US_03_Mesh_ArmStandard
US_03_Mesh_beard
US_03_Mesh_Boots
US_03_Mesh_Buckles
US_03_Mesh_eyeball
US_03_Mesh_Eyelashes
US_03_Mesh_Face
US_03_Mesh_Gloves
US_03_Mesh_Headgear
US_03_Mesh_Kits
US_03_Mesh_LegStandard
US_03_Mesh_MetalRing
US_03_Mesh_MOLLE
US_03_Mesh_Pads
US_03_Mesh_Pouches
US_03_Mesh_Straps
Database Explorer
Terrain control panel
Properties
Inputs
AlphaInColormapEnable
Camo 0,5/0,5/0,5
Color 1/1/1/1
Detail 0,5/0,5/0,5
Dirt 1/1/1/1
FresnelBias 0,1
FresnelExponent 2
Normal 0/0/0
NormalBlend 0,5
NormalDetail 0/0/0
Occlusion 1/1/1
Opacity 1
roughness 0,7
Specular 0,08/0,08/0,08
Type: InstanceShaderNode
US_03_Mesh_Pads
US_03_Mesh_MOLLE*
Clothing2
normalmap
detailmap
normaldetailmap
colormap
occlusionTexture
dirtmap
Blend
Normalize
Color
Multiply2
MixOut
texCoord3
texCoord2
texCoord4
AlphaInColormapEnable
Camo
Color
Detail
Dirt
FresnelBias
FresnelExponent
Normal
NormalBlend
NormalDetail
Occlusion
Opacity
roughness
Specular
Clothing3
Output
UpdateIndex(): 00:00:00.1249968
OpenIndex(): 00:00:08.2029150
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' modified.
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' saved
Partition 'Characters/US/US_Models/Shaders/US_03_Mesh_MOLLE' modified.

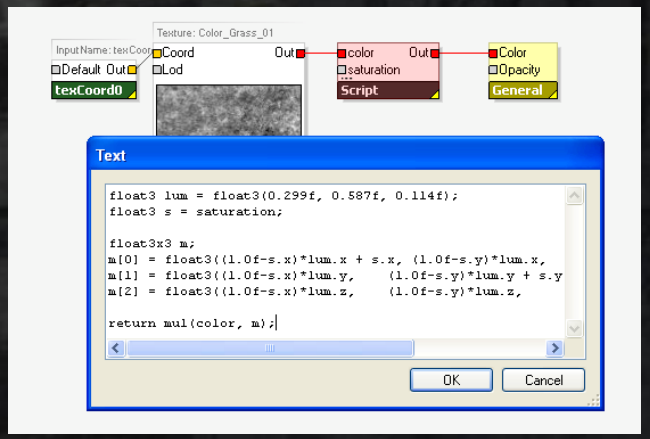
Surface shader nodes

- Built-in nodes
  - Basic arithmetic (mul, add, divide)
  - Geometric (fresnel, refraction)
  - Logical (platform, or, side, conditional)
  - Parameters (scalar, vec2, vec4, bool)
  - Values (position, z, normal, eye vector)
  - Lighting (phong, sub-surface)
  - Root (general, offset, multi output)
  - Misc (curve, script, parallax offset)



Surface shader complexity

- Tedious to create arithmetic-heavy shaders as graphs
  - Requires lots small nodes with connections between everything
  - = Spaghetti shaders



- Script nodes can help
  - Have arbitrary number of inputs and outputs
  - Write HLSL function to process input to output
  - Similar to how the shader pipeline works internally

Surface shader complexity (cont.)

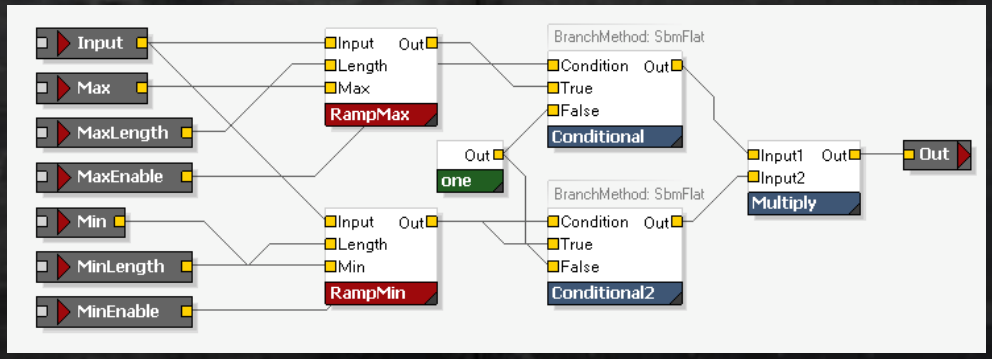
- Lots of people work with surface shaders
  - Rendering programmers, technical/lead artists, artists, outsourcing
- Not everybody want/need/can create shaders fully from scratch
  - Should be able to work on the level most suited
- Custom shaders on everything is bad
  - Quality, maintenance, performance
- But the ability to create custom shaders is good
  - Experimentation, pre-production, optimization

Shader complexity solutions

- Settle on a resonable middle-ground
  - Common approach
  - Most likley artist-centric
  - Programmers mostly work on the code level instead and expose new nodes
  - Not as scaleable
- Directly support authoring at multiple levels
  - More complex
  - But exactly what we want

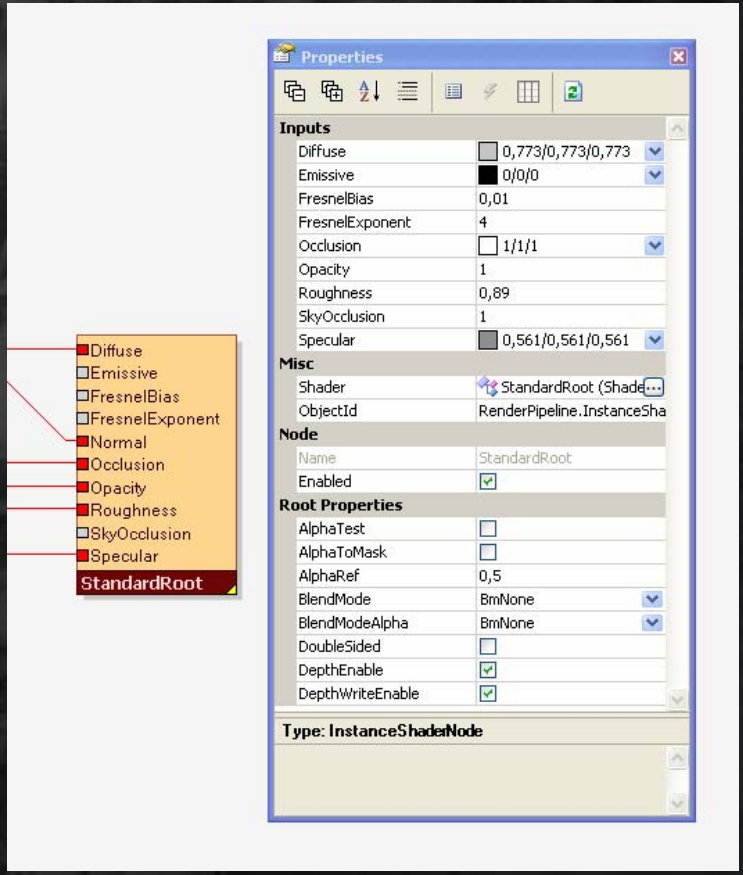
Instance shaders

- Our solution
- An instance shader is a graph network that can be instanced as a node in another shader
  - Think C++ functions
- Hide and encapsulate functionality on multiple levels by choosing inputs & outputs to expose
- Heavily used in BFBC



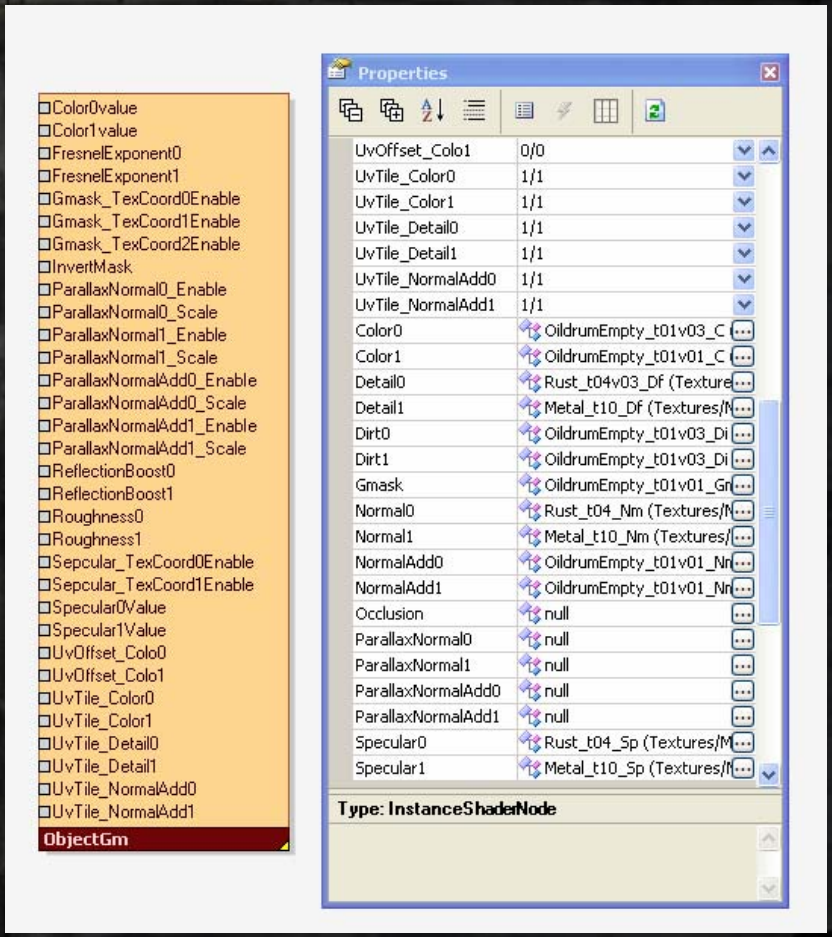
StandardRoot instance shader

- Programmer created
- Phong BRDF
- Basic inputs for diffuse, specular, emissive, fresnel & occlusion
- Transparency properties
- Base for 90% of our shaders



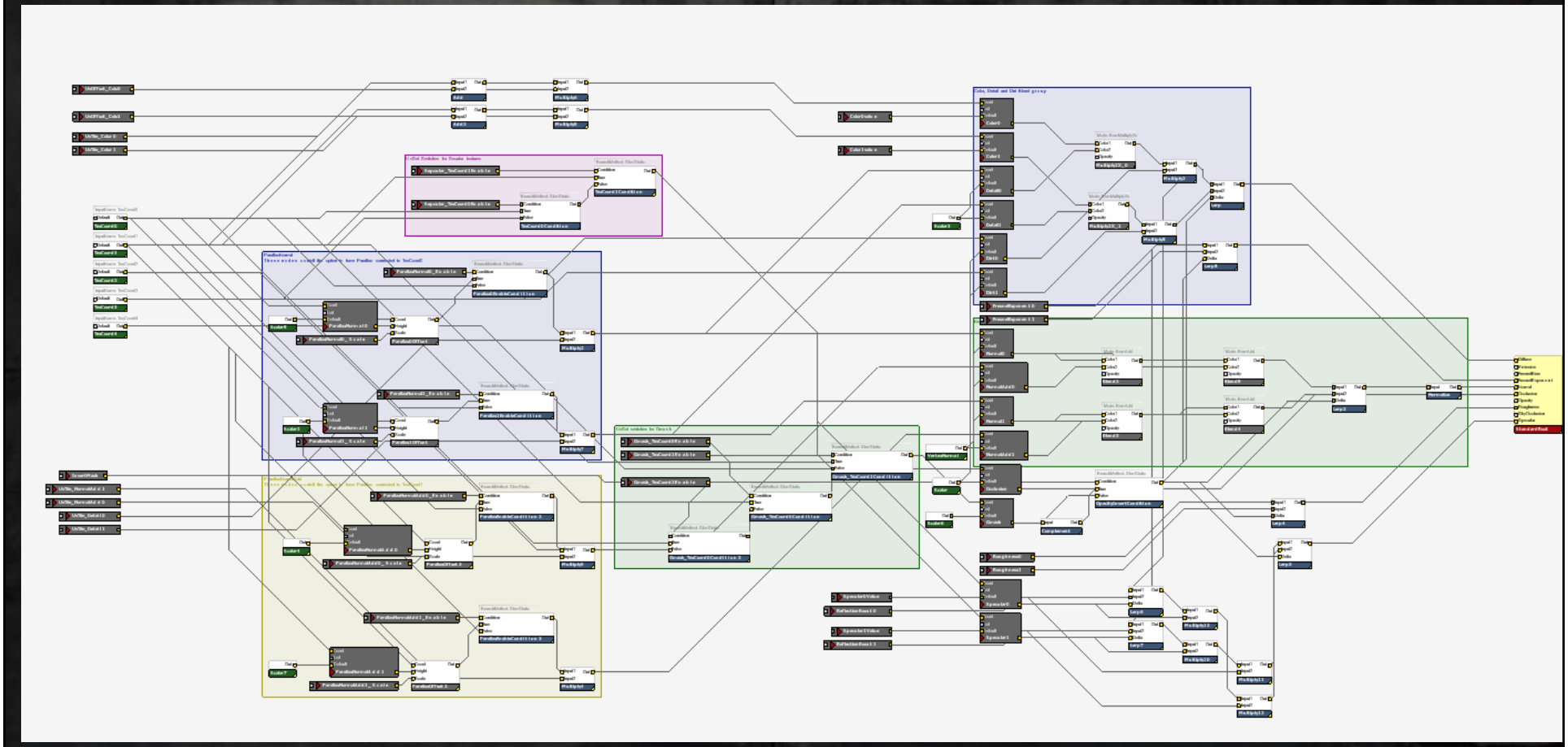
ObjectGm instance shader

- Artist created
- Locked down shader for objects
- Very general, lots of features in same shader
- Many properties instead of inputs



screenshot, node shou[illegible] few inputs

Inside ObjectGm shader



DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Shading system pipeline

- Big complex offline pre-processing system
  - Systems report wanted state combinations
- Generates shading solutions for runtime
  - Solution for each shading state combination
  - Example: A mesh with stream instancing, a surface shader, light-scattering and affected by a outdoor light source & shadow and 2 point lights for Xbox 360
- Generates HLSL vertex & pixel shaders
- Solutions contains complete state setup
  - Passes, shaders, constants, parameters, textures..

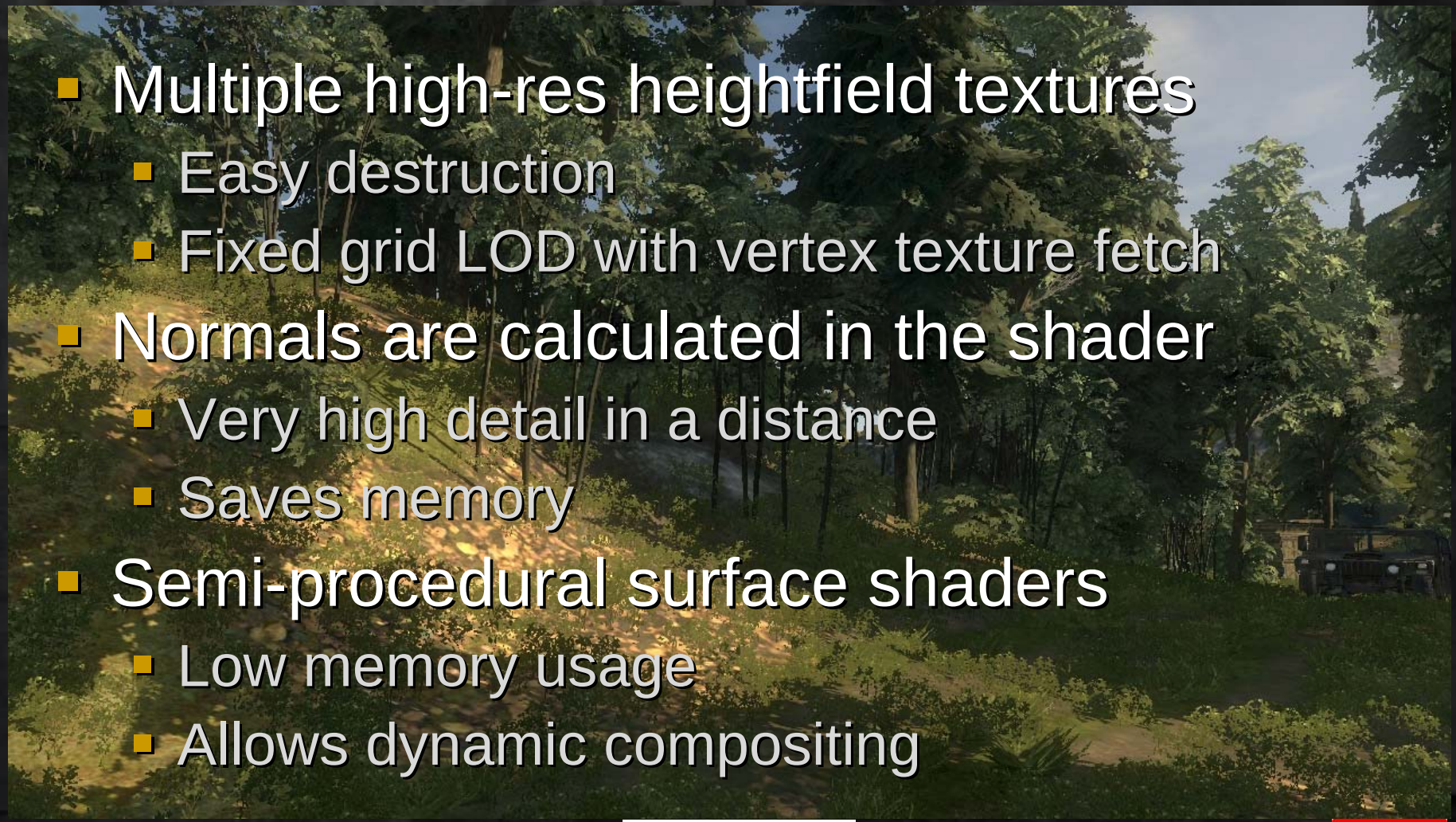
Shading system runtime

- User queues up *render blocks*
  - Geometry & high-level state combinations
- Looks up solutions for the state combinations
  - Pipeline created these offline
- Blocks dispatched by backend to D3D/GCM
  - Blocks are sorted (category & depth)
  - Backend sets platform-specific states and shaders
    - Determined by pipeline for that solution
    - Thin & dumb
  - Draw

Terrain

- Important for many of our games
  - Rallisport & Battlefield series
- Goals
  - Long view distance with true horizon
    - 32x32 km visible, 2x2 – 4x4 playable
  - Ground destruction
  - High detail up close and far away
  - Artist control
  - Low memory usage

Terrain (cont.)



- Multiple high-res heightfield textures
  - Easy destruction
  - Fixed grid LOD with vertex texture fetch
- Normals are calculated in the shader
  - Very high detail in a distance
  - Saves memory
- Semi-procedural surface shaders
  - Low memory usage
  - Allows dynamic compositing

Procedural shader splatting

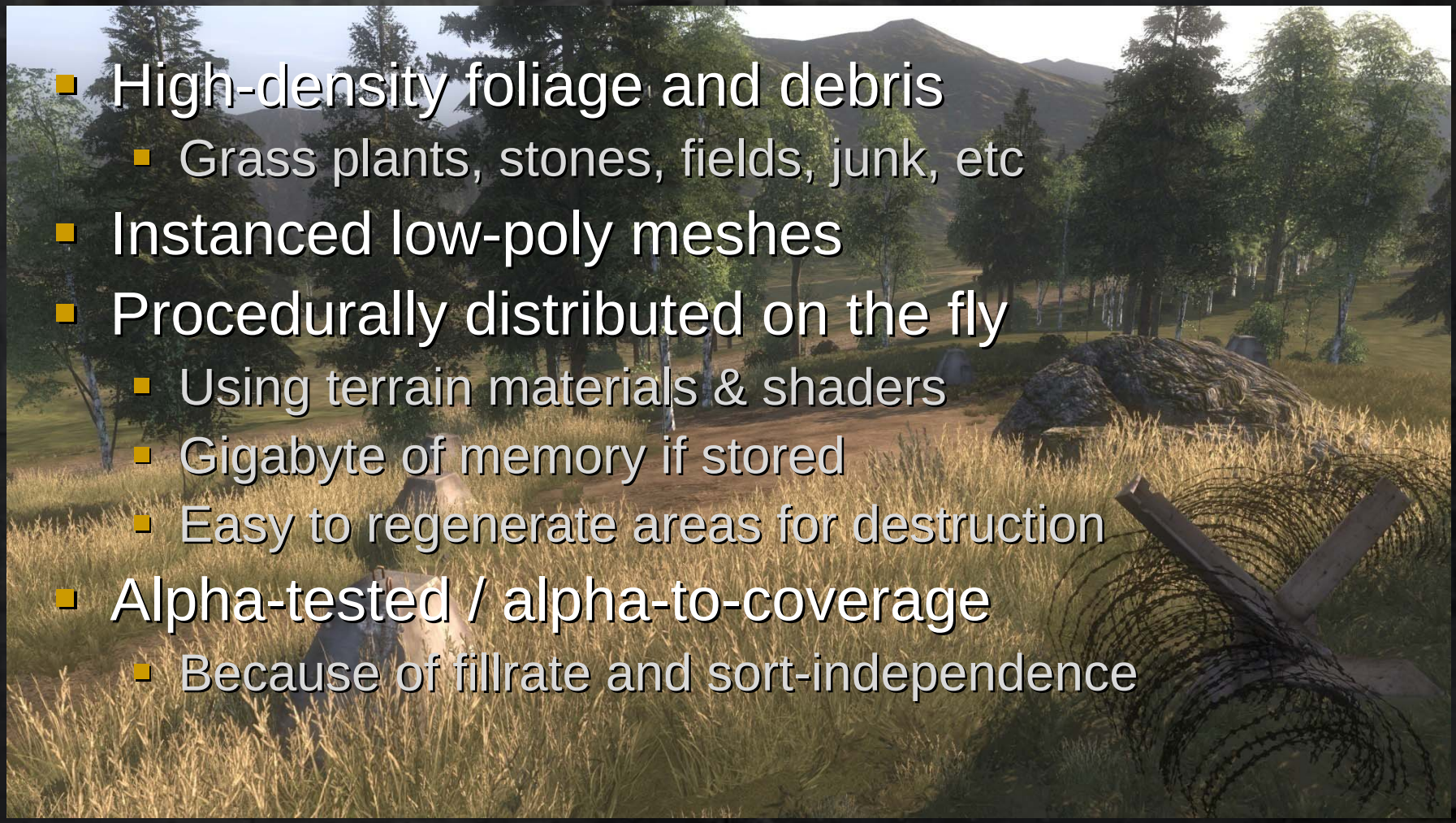
- Surface shaders for each material
  - Access to per-pixel height, slope, normal, sparse mask textures & decals
  - Abitrary texture compositing & blending
- Material shaders are merged and blended
  - For each material combination
  - Heavy single-pass shaders
  - Lots of dynamic branching
- Very flexible & scaleable
- More details at Siggraph'07 course

Without undergrowth

With undergrowth

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Undergrowth



- High-density foliage and debris
  - Grass plants, stones, fields, junk, etc
- Instanced low-poly meshes
- Procedurally distributed on the fly
  - Using terrain materials & shaders
  - Gigabyte of memory if stored
  - Easy to regenerate areas for destruction
- Alpha-tested / alpha-to-coverage
  - Because of fillrate and sort-independence

Undergrowth generation

- Patches are dynamically allocated around camera
- When patches become visible or is changed
  - GPU renders 8-12 material visibility values, terrain normal and cached textures
  - PPU/SPU processes textures and pseudo-randomly distributes mesh instances within patch
- Easy rendering after generation
  - Arbitrary meshes and surface shaders can be used
  - Rendered with standard stream instancing
  - Only visual, no collision
- Perfect fit for D3D10 Stream Output
  - Keeps everything on GPU, reduces latency

Outline

- Introduction
- Frostbite Engine
- **Examples from demos**
- Conclusions

Practical Example: Mountains Generation and Realistic Snow Accumulation

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

Use fBm to Generate Mountain Terrain

- Compute multiple octaves (10-50) of fBm noise to use as displacement
  - Vertex texture-based displacement
- Variety of options
  - Compute displacement directly in the shader per frame
    - Great for animating earthquakes
  - Stream out and reuse as necessary
  - Precompute for static geometry
- Use masks to vary noise computation / parameters as needed

Mountains: Wireframe

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

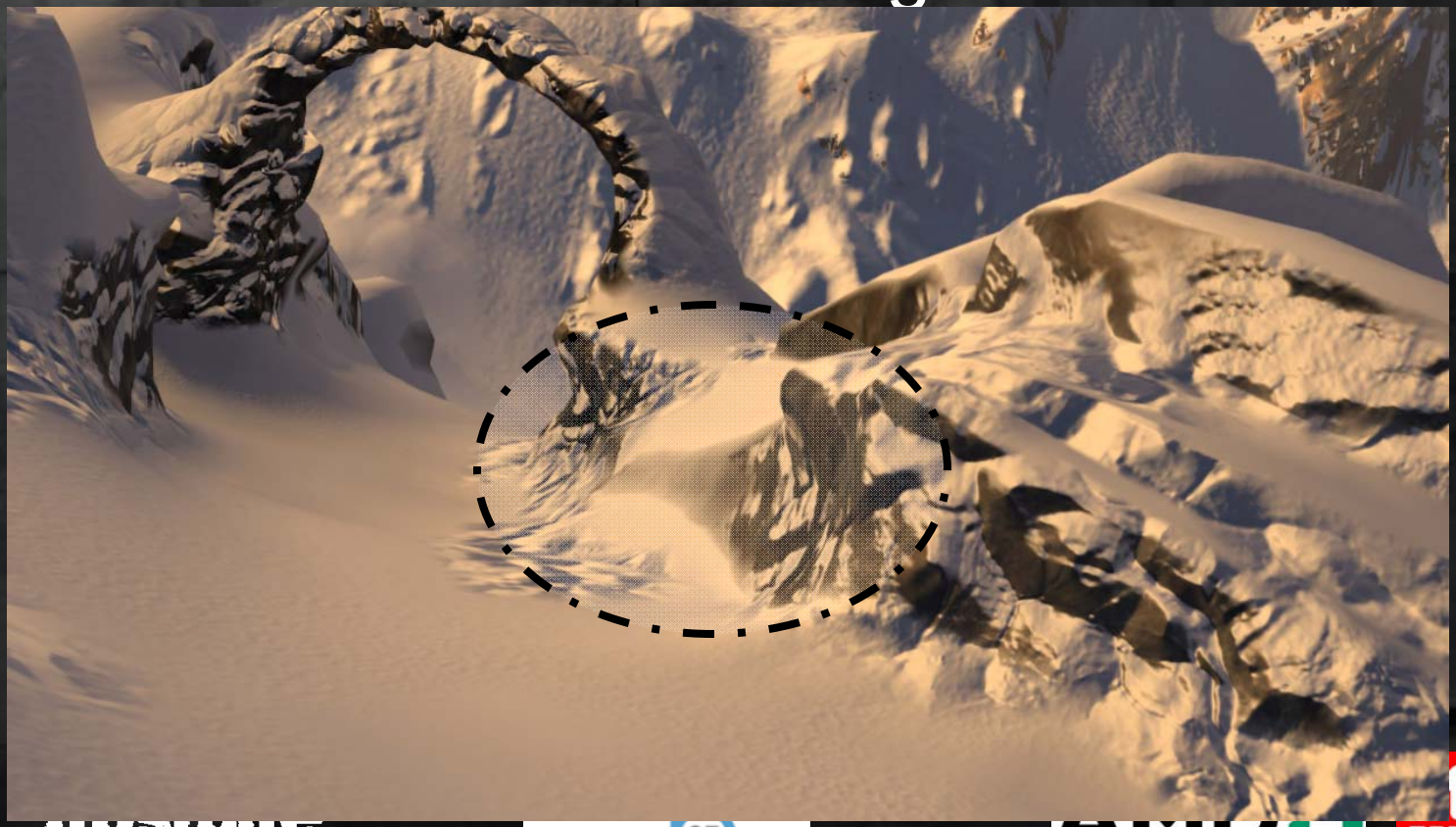
Controlling Snow Accumulation

- Want snow accumulation to correlate to the objects - automatically
- Determine snow coverage procedurally
- Idea: use the combination of the geometric normal and the bump map normal to control snow coverage
  - With blending factors which control how we "accumulate"
    or "melt" snow
  - i.e. its appearance on the geometry (Eg: Mountain)
  - Depending on the geometric normal orientation


frostbite
GameDevelopers®
Conference
07
AMD
ATI
RADEON
GRAPHICS

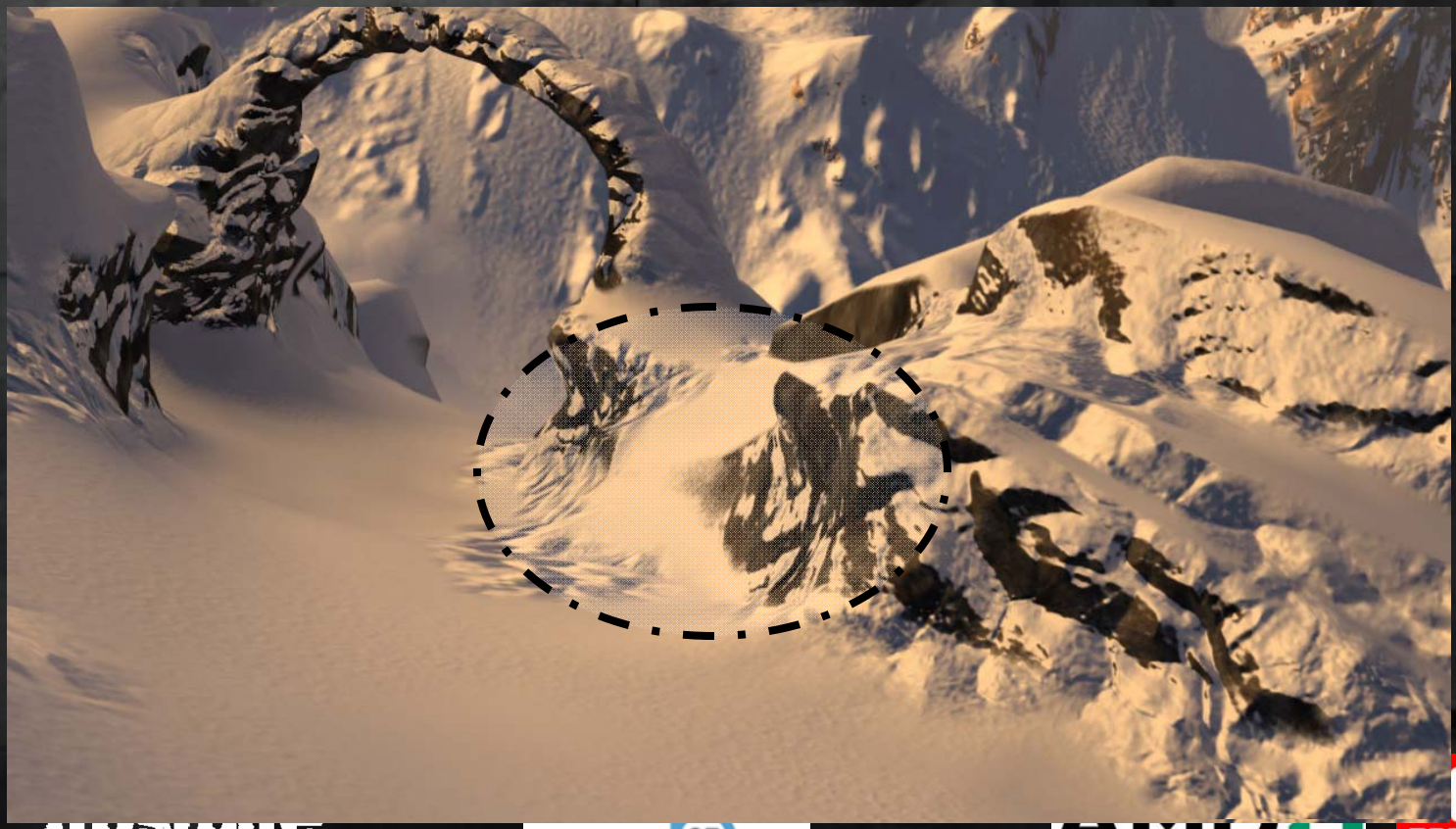
What If We Don't Use Noise?

- Straight-forward blend creates a sharp crease between snow and ground



Break Up the Monotony

- Use noise to adjust the blend between snow and rock for a natural transition



Demo

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS

If You Want to Know More...

- About generating noise on the GPU
- Different types of procedural noise
- And more snow accumulation
  - GDC “The Importance of Being Noisy: Fast, High Quality Noise”, N. Tatarchuk
    - On AMD developer website

Other Procedural Techniques

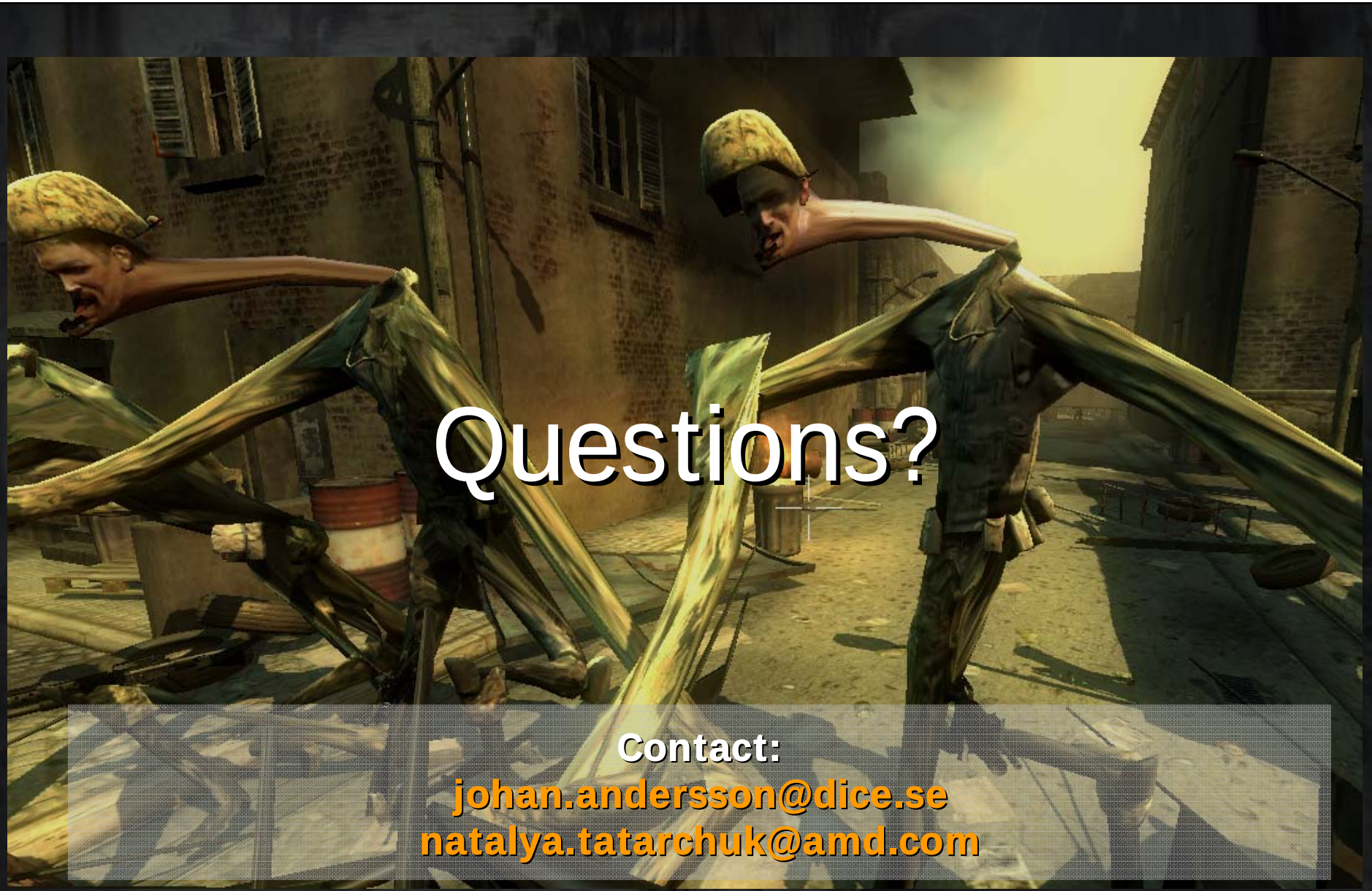
- Procedural Tools and Techniques for Current and Future Game Platforms

  by Jeremy Shopf (AMD) and

  Sebastien Deguy (Allegorithmic)

Thanks!

- Chris Oat & Abe Wiley (snowy mountains)



Contact:
johan.andersson@dice.se
natalya.tatarchuk@amd.com

DICE frostbite GameDevelopers® Conference 07 AMD ATI Radeon GRAPHICS